

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-099428

(43)Date of publication of application : 05.04.2002

(51)Int.Cl.

G06F 9/45

(21)Application number : 2001-201414

(71)Applicant : CANON INC

(22)Date of filing : 02.07.2001

(72)Inventor : JOHN CHARLES BROOKE

(30)Priority

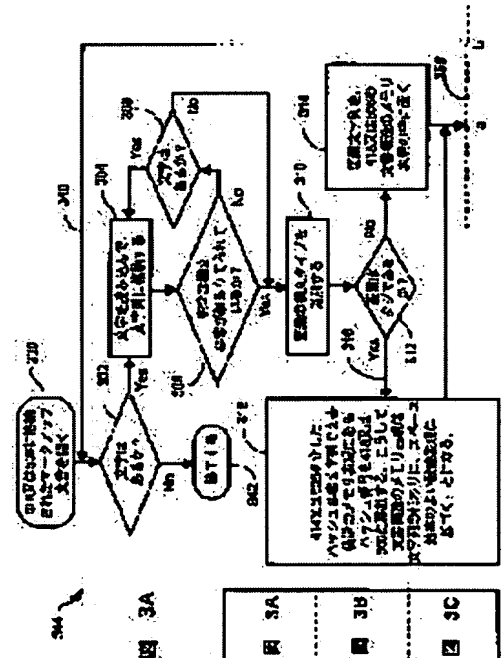
Priority number : 2000 PQ8495 Priority date : 30.06.2000 Priority country : AU

(54) HASH COMPACT XML PARSER

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a method of parsing a markup language document including syntactic elements to improve a competing with a memory and a processing burden in an apparatus having hardware constrains.

SOLUTION: A hash compact XML parser comprises a step 310 of identifying a type of an element for one of the syntactic elements, a step 318 of processing the element by determining a hash representation thereof if the type is a first type and a step 314 of augmenting an at least partial structural representation of the document using the hash representation if the type is the first type.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号
特開2002-99428
(P2002-99428A)

(43)公開日 平成14年4月5日(2002.4.5)

(51)Int.Cl. ⁷	識別記号	F I	テーマコード [*] (参考)
G 0 6 F 9/45		G 0 6 F 9/44	3 2 2 B 5 B 0 8 1 3 2 2 C

審査請求 未請求 請求項の数58 O L 外国語出願 (全100頁)

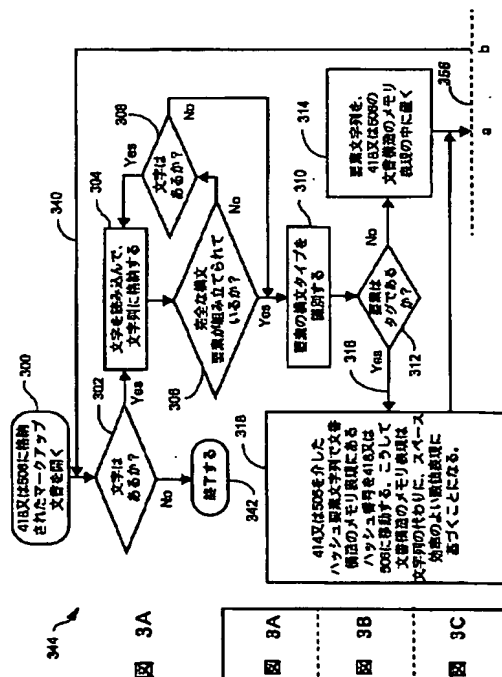
(21)出願番号	特願2001-201414(P2001-201414)	(71)出願人	000001007 キヤノン株式会社 東京都大田区下丸子3丁目30番2号
(22)出願日	平成13年7月2日(2001.7.2)	(72)発明者	ジョン チャールズ ブルック オーストラリア国 2113 ニュー サウス ウェールズ州、 ノース ライド、 ト ーマス ホルト ドライブ 1 キヤノン インフォメーション システムズ リサ ーチ オーストラリア プロプライエタリ ー リミテッド 内
(31)優先権主張番号	P Q 8 4 9 5	(74)代理人	100076428 弁理士 大塚 康徳 (外3名) Fターム(参考) 5B081 AA10 CC11 CC61
(32)優先日	平成12年6月30日(2000.6.30)		
(33)優先権主張国	オーストラリア (AU)		

(54)【発明の名称】 ハッシュコンパクトXMLパーサ

(57)【要約】 (修正有)

【課題】 ハードウェア制約のある装置でのメモリとの競合および処理負担を改善する構文要素を含むマークアップ言語文書の解析方法を提供する。

【解決手段】 構文要素の1つに対し、その要素のタイプを識別し310、前記タイプが第1タイプの場合はハッシュ表現を決定することによってその要素を処理し318、前記タイプが前記第1タイプの場合は、ハッシュ表現を用いたその文書の少なくとも部分的な構造表現を強める314、という手続きを含む。



【特許請求の範囲】

【請求項1】構文要素を含むマークアップ言語文書を解析する解析方法であって、前記構文要素の内の一要素に対し、

前記一要素のタイプを識別する識別ステップと、

前記タイプが第1タイプであれば、そのハッシュ表現を決定することによってその要素を処理する処理ステップと、

前記タイプが前記第1タイプであれば、前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、を含むことを特徴とする解析方法。

【請求項2】イベント駆動型であることを特徴とする請求項1に記載の解析方法。

【請求項3】ハッシュアルゴリズム、関連ユニバーサルリファレンスインディケータに依存する前記ハッシュアルゴリズムへの第1リファレンス、関連名前空間に依存する前記ハッシュアルゴリズムへの第2リファレンス、及び、関連エクステンドマークアップ言語宣言に依存する前記ハッシュアルゴリズムへの第3リファレンス、のいずれかを用いて前記ハッシュ表現を決定することを特徴とする請求項1に記載の解析方法。

【請求項4】前記第1タイプは、構造要素或はその一部、前記構造要素の定義、前記構造要素の宣言、及び前記構造要素のマッチ、のいずれかであることを特徴とする請求項2に記載の解析方法。

【請求項5】前記構造要素はタグであることを特徴とする請求項4に記載の解析方法。

【請求項6】前記ハッシュ表現は、前記一つの構文要素ごとのユニークなコードであり、前記一構文要素は第1文字数以下であることを特徴とする請求項2に記載の解析方法。

【請求項7】前記ハッシュ表現は、前記一つの構文要素ごとにユニークなコードではなく、前記一構文要素は、(i) 前記要素の文字数、及び(ii) 前記要素の文字の順列の許容数、のうちの少なくとも一つの観点から、ある可能性レベルまで制約を受けることを特徴とする請求項2に記載の解析方法。

【請求項8】前記コードは、英数字を含むことを特徴とする請求項6に記載の解析方法。

【請求項9】前記処理ステップは、(i) 前記第1タイプの第1インスタンスである前記一構文要素と(ii) 前記第1タイプの第2インスタンスである他方の構文要素と、の両方の拡張ハッシュ表現を決定するサブステップを含み、前記第1インスタンス内に、前記第2インスタンスがネストになっていることを特徴とする請求項2に記載の解析方法。

析方法。

【請求項10】前記構文要素の内の他要素について、更に、

前記他要素のタイプを識別するステップと、

前記他要素のタイプが前記第1タイプと同等である場合に、

(i) 前記他要素を処理し、それによってその第2ハッシュ表現を決定する第2処理ステップと、

(ii) 前記第2ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、を有し、

前記処理ステップ及び前記第2処理ステップは、前記一要素と前記他要素の間に第1の関係が存在する場合に、第1の関係に対応する第2の関係が、前記一要素のハッシュ表現と、前記他要素のハッシュ表現との間に存在することを特徴とする請求項1に記載の解析方法。

【請求項11】前記一要素は、開始タグであり、前記他要素は、終了タグであり、前記一要素のハッシュ表現は、対応ハッシュ開始タグであり、前記他要素の第2ハッシュ表現は、対応ハッシュ終了タグであることを特徴とする請求項10に記載の解析方法。

【請求項12】前記終了タグは、前記開始タグの第1変形であり、前記ハッシュ終了タグは、前記ハッシュ開始タグの第2変形であり、前記第2変形は、前記第1変形に対応することを特徴とする請求項11に記載の解析方法。

【請求項13】前記終了タグは、そこに識別文字が組込まれること以外は、前記開始タグと同一であり、前記ハッシュ終了タグは、前記ハッシュ開始タグと同一のもの、そこに識別文字が組込まれること以外は前記ハッシュ開始タグと同一のもの、及びオペレータによって処理されたハッシュ開始タグと同一のもの、

の内の少なくともの一つであることを特徴とする請求項12に記載の解析方法。

【請求項14】前記一要素及び前記他要素は、それぞれ、開始タグ及び終了タグを含み、第1タグペアをなし、

前記第1タグペアの為の対応ハッシュ開始タグ及び対応ハッシュ終了タグは、前記文書の前記部分的構造表現に組込まれ、

前記文書は、更に、開始タグと終了タグをそれぞれ含む第2タグペアを含み、該第2タグペアは、前記文書の前記第1タグペア内にネストされ、

前記解析方法は、更に、前記第2タグペアを処理し、対応第2ハッシュ開始／終

アタグを形成するステップと、

前記第2ハッシュ開始／終了タグが、前記第1タグペアの為の前記ハッシュ開始／終了タグとの関係で、前記第1タグペア内での前記第2タグペアのネストと同等のネストを示すため、前記対応第2ハッシュ開始／終了タグを用いて、前記文書の前記少なくとも部分的構造表現を増大する増大ステップと、
を有することを特徴とする請求項12に記載の解析方法。

【請求項15】前記増大ステップの前に、更に、
第1ハッシュ開始タグを第2ハッシュ開始タグと連結し、第1ハッシュ終了タグを第2ハッシュ終了タグと連結し、それにより、前記第2タグペアについてのそれぞれの拡張ハッシュ開始／終了タグを形成し、
前記増大ステップは、前記第2タグペアについての前記それぞれの拡張ハッシュ開始／終了タグを用いて実行され、

前記拡張ハッシュ開始／終了タグは、前記第1タグペアの前記ハッシュ開始／終了タグとの関係で、前記第1タグペア内での前記第2タグペアのネストと同等のネストを示すことを特徴とする請求項14に記載の解析方法。

【請求項16】前記増大ステップに続いて、構文規則に対する整形性チェックステップが行われ、
該整形性チェックは、前記マークアップ言語文書の前記少なくとも部分的な構造表現における要素の対応ハッシュ表現を数的に比較することによって、前記マークアップ言語文書の前記少なくとも部分的な構造表現を構文規則に照らしてチェックするステップを含むことを特徴とする請求項1に記載の解析方法。

【請求項17】前記数的な比較のステップに続いて、更に、
前記構文規則に従い、前記第1タイプではない要素の対応非ハッシュ表現を、文字列比較するステップを行うことを特徴とする請求項16に記載の解析方法。

【請求項18】前記第1タイプは、
構造要素或はその一部、
前記構造要素の定義、
前記構造要素の宣言、及び前記構造要素のマッチ、
のいずれかであることを特徴とする請求項1に記載の解析方法。

【請求項19】前記構造要素はタグであることを特徴とする請求項18に記載の解析方法。

【請求項20】前記文書の前記少なくとも部分的な構造表現の整形性を、構文規則に照らしてチェックするチェックステップを更に含むことを特徴とする請求項14に記載の解析方法。

【請求項21】前記構文規則は、タグの適正なネストに関するものであり、
前記チェックステップは、
前記文書の前記少なくとも部分的な構造表現におけるハ

ッシュタグに亘る数的比較を行い、それにより、前記第1ハッシュ開始／終了タグと、前記第2ハッシュ開始／終了タグを識別するサブステップと、

前記第2ハッシュ開始／終了タグが前記第1ハッシュ開始／終了タグとの関係で適正なネストを示すことを確認するサブステップと、

を含むことを特徴とする請求項20に記載の解析方法。

【請求項22】前記数的比較に続いて、
前記構文規則に従い、前記文書の前記少なくとも部分的な構造表現におけるそれぞれのタグの非ハッシュ部分に亘り、文字列比較を行うステップを更に含むことを特徴とする請求項21に記載の解析方法。

【請求項23】構文規則に照らして、前記文書の前記少なくとも部分的な構文表現の整形性をチェックするステップを更に有することを特徴とする請求項15に記載の解析方法。

【請求項24】前記構文規則は、タグの適正なネストに関するものであり、

前記チェックステップは、

前記文書の前記少なくとも部分的な構造表現におけるハッシュタグに亘り数的比較を行い、それにより、前記第1ハッシュ開始／終了タグと、前記第2ハッシュ開始／終了タグを識別するサブステップと、

前記拡張ハッシュ開始／終了タグが、前記第1ハッシュ開始／終了タグとの関係で適正なネストを示すことを確認するサブステップと、

を含むことを特徴とする請求項23に記載の解析方法。

【請求項25】前記数的比較に続いて、
前記文書の前記少なくとも部分的な構造表現におけるそれぞれのタグの非ハッシュ部分にわたり、文字列比較を行うステップを更に含むことを特徴とする請求項24に記載の解析方法。

【請求項26】前記整形性チェックステップは、妥当性参照文書VRDに照らす妥当性検査ステップによって、
(a) 続けられ、(b) 含まれ、又は(c) 置き換えられ、

前記妥当性検査ステップは、

(a) 前記VRDを処理する処理サブステップを含み、
該処理サブステップは、VRDにおけるの構文要素ごとに、

(i) VRDの前記構文要素のタイプを識別するサブサブステップと、

(ii) 前記タイプが、前記第1タイプの場合に、前記構文要素を、そのハッシュ表現を決定することによって処理するサブサブステップと、を含み、

前記妥当性検査ステップは、更に、

(b) 処理されたVRDに照らして前記マークアップ言語文書の前記少なくとも部分的な構造表現をチェックするチェックサブステップを含み、該チェックサブステップは、要素の対応ハッシュ表現を数的に比較するサブサ

ブステップを含むことを特徴とする請求項 16 に記載の解析方法。

【請求項 27】VRD に照らしてマークアップ言語文書の妥当性検査を行う検査方法であって、

(a) マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第 1 ドキュメントタグでなければ、処理するステップを有し、該処理は、

(i) 前記ドキュメントタグの階層位置を決定するステップと、

(i i) 前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するステップと、

(i i i) 前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するステップと、を含み、

前記検査方法は更に、

(b) VRD を、そこで識別されるタグごとに、該タグが対応タグ階層の第 1 タグでない場合に、処理するステップを有し、該処理は、

(i) 前記タグの階層位置を決定するステップと、

(i i) 対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するステップと、

(i i i) 前記タグの前記拡張ハッシュ表現をリストに格納するステップと、を含み、

前記検査方法は更に、

(c) 前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバーの妥当なサブセットである場合には、前記マークアップ言語文書の妥当性を検査するステップを有することを特徴とする検査方法。

【請求項 28】VRD に照らしてマークアップ言語文書の妥当性を検査する検査方法であって、

(a) 前記 VRD を、そのなかで識別された構造要素ごとに、処理する処理ステップを含み、該処理ステップは、

(i) 前記構造要素の構文属性を決定するステップと、

(i i) 前記構造要素のハッシュ表現を決定するステップと、

(i i i) 前記 VRD の構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するステップと、を含み、

前記検査方法は更に、

(b) マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理する処理ステップを含み、該処理ステップは、

(i) 前記文書構造要素の構文属性を決定するステップ

と、

(i i) 前記文書構造要素のハッシュ表現を決定するステップと、

(i i i) 前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するステップと、を含み、

前記検査方法は更に、

(c) 前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記 VRD の前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するステップを有することを特徴とする検査方法。

【請求項 29】前記数値比較ステップに続いて、更に、前記第 1 タイプでない要素の対応非ハッシュ表現の文字列比較ステップを行うことを特徴とする請求項 26 に記載の解析方法。

【請求項 30】前記第 1 タイプは、

構造要素或はその一部、

前記構造要素の定義、

前記構造要素の宣言、

前記構造要素のマッチ、

のいずれかであることを特徴とする請求項 26 に記載の解析方法。

【請求項 31】前記構造要素はタグであることを特徴とする請求項 30 に記載の解析方法。

【請求項 32】構文要素を含むマークアップ言語文書を符号化する符号化方法であって、

前記構文要素の一つについて、

構文要素のタイプを識別するステップと、

(i) 前記タイプが第 1 タイプであれば、前記構文要素のハッシュ表現を決定すること、

(i i) 前記タイプが第 1 タイプでなければ、前記構文要素の圧縮表現を決定すること、

(i i i) 前記構文要素を保持すること、

のいずれかによって、前記構文要素を処理するステップと、

を含むことを特徴とする符号化方法。

【請求項 33】符号化構文要素を含むマークアップ言語文書を復号する復号方法であって、

前記符号化構文要素の一つについて、

前記符号化構文要素のタイプを識別するステップと、

(i) 前記タイプが第 1 タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、

(i i) 前記タイプが第 1 タイプでなければ、前記符号化構文要素の解凍表現を決定すること、

(i i i) 前記符号化構文要素を保持すること、

の少なくとも一つで、前記符号化構文要素を処理するステップと、

を含むことを特徴とする復号方法。

【請求項 34】構文要素を含むマークアップ言語文書を

解析する解析装置であって、
 前記構文要素のタイプを識別する識別手段と、
 前記タイプが第1タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理する処理手段と、
 前記タイプが第1タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大する増大手段と、
 を含むことを特徴とする解析装置。

【請求項35】VRDに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、

(a) マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第1ドキュメントタグでなければ、処理する手段を有し、該手段は、

(i) 前記ドキュメントタグの階層位置を決定する手段と、

(ii) 前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定する手段と、

(iii) 前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納する手段と、を含む、

前記検査装置は更に、

(b) VRDを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理する手段を有し、該手段は、

(i) 前記タグの階層位置を決定する手段と、

(ii) 対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定する手段と、

(iii) 前記タグの前記拡張ハッシュ表現をリストに格納する手段と、を含む、

前記検査装置は更に、

(c) 前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバーの適当なサブセットであるかのどちらかであることを立証し、それによって、前記マークアップ言語文書の妥当性を検査する手段を有することを特徴とする検査装置。

【請求項36】VRDに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、

(a) 前記VRDを、そのなかで識別された構造要素ごとに、処理する手段を含み、該手段は、

(i) 前記構造要素の構文属性を決定する手段と、

(ii) 前記構造要素のハッシュ表現を決定する手段と、

(iii) 前記VRDの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納する手段と、

を含み、

前記検査装置は更に、

(b) マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理する手段を含み、該手段は、

(i) 前記文書構造要素の構文属性を決定する手段と、

(ii) 前記文書構造要素のハッシュ表現を決定する手段と、

(iii) 前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納する手段と、を含む、

前記検査装置は更に、

(c) 前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現と、前記VRDの前記構造表現における対応構文属性及びハッシュ表現とを比較し、それにより、前記マークアップ言語文書の妥当性を検査する手段を有することを特徴とする検査装置。

【請求項37】構文要素を含むマークアップ言語文書を符号化し、前記文書の少なくとも部分的構造表現を形成する符号化装置であって、

構文要素のタイプを識別する手段と、

(i) 前記タイプが第1タイプであれば、前記構文要素のハッシュ表現を決定すること、

(ii) 前記タイプが第1タイプでなければ、前記構文要素の圧縮表現を決定すること、

(iii) 前記構文要素を保持すること、

のいずれかによって、前記構文要素を処理する手段と、を含むことを特徴とする符号化装置。

【請求項38】符号化構文要素を含むマークアップ言語文書を復号する復号装置であって、

前記符号化構文要素のタイプを識別する手段と、

(i) 前記タイプが第1タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、

(ii) 前記タイプが第1タイプでなければ、前記符号化構文要素の解凍表現を決定すること、

(iii) 符号化構文要素を保持すること、

の少なくとも一つによって、前記符号化構文要素を処理する手段と、

を含むことを特徴とする復号装置。

【請求項39】コンピュータに構文要素を含むマークアップ言語文書を解析する手続を実行させるコンピュータプログラムであって、

前記構文要素のタイプを識別するコードと、

前記タイプが第1タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、

前記タイプが第1タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、

を含むことを特徴とするコンピュータプログラム。

【請求項40】コンピュータにVRDに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムであって、

(a) マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第1ドキュメントタグでなければ、処理するコードを有し、該コードは、

(i) 前記ドキュメントタグの階層位置を決定するコードと、

(i i) 前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、

(i i i) 前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、

前記コンピュータプログラムは更に、

(b) VRDを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理するコードを有し、該コードは、

(i) 前記タグの階層位置を決定するコードと、

(i i) 対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、

(i i i) 前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、

前記コンピュータプログラムは更に、

(c) 前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合には、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム。

【請求項41】コンピュータにVRDに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムであって、

(a) 前記VRDを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、

(i) 前記構造要素の構文属性を決定するコードと、

(i i) 前記構造要素のハッシュ表現を決定するコードと、

(i i i) 前記VRDの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、

前記プログラムは更に、

(b) マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、

(i) 前記文書構造要素の構文属性を決定するコードと、

(i i) 前記文書構造要素のハッシュ表現を決定するコードと、

(i i i) 前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、

前記プログラムは更に、

(c) 前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記VRDの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム。

【請求項42】コンピュータに構文要素を含むマークアップ言語文書を符号化する手続を実行させるコンピュータプログラムであって、

構文要素のタイプを識別するコードと、

(i) 前記タイプが第1タイプであれば、前記構文要素のハッシュ表現を決定すること、

(i i) 前記タイプが第1タイプでなければ、前記構文要素の圧縮表現を決定すること、

(i i i) 前記構文要素を保持すること、

のいずれかによって、前記構文要素を処理するコードと、

を含むことを特徴とするコンピュータプログラム。

【請求項43】コンピュータに符号化構文要素を含むマークアップ言語文書を復号する手続を実行させるコンピュータプログラムであって、

前記符号化構文要素のタイプを識別するコードと、

(i) 前記タイプが第1タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、

(i i) 前記タイプが第1タイプでなければ、前記符号化構文要素の解凍表現を決定すること、

(i i i) 符号化構文要素を保持すること、

の少なくとも一つで、前記符号化構文要素を処理するコードと、

を含むことを特徴とするコンピュータプログラム。

【請求項44】コンピュータに構文要素を含むマークアップ言語文書を解析する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、

前記コンピュータプログラムは、

前記構文要素のタイプを識別するコードと、

前記タイプが第1タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、

前記タイプが第1タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、

を含むことを特徴とするコンピュータプログラム製品。

【請求項45】コンピュータにVRDに照らしてマーク

アップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、

前記コンピュータプログラムは、

(a) マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第1ドキュメントタグでなければ、処理するコードを有し、該コードは、

(i) 前記ドキュメントタグの階層位置を決定するコードと、

(i i) 前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、

(i i i) 前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、

前記コンピュータプログラムは更に、

(b) VRDを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理するコードを有し、該コードは、

(i) 前記タグの階層位置を決定するコードと、

(i i) 対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、

(i i i) 前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、

前記コンピュータプログラムは更に、

(c) 前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム製品。

【請求項46】コンピュータにVRDに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、

前記プログラムは、

(a) 前記VRDを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、

(i) 前記構造要素の構文属性を決定するコードと、

(i i) 前記構造要素のハッシュ表現を決定するコードと、

(i i i) 前記VRDの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、

前記プログラムは更に、

(b) マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、

該コードは、

(i) 前記文書構造要素の構文属性を決定するコードと、

(i i) 前記文書構造要素のハッシュ表現を決定するコードと、

(i i i) 前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、

前記プログラムは更に、

(c) 前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記VRDの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム製品。

【請求項47】構文要素を含むマークアップ言語文書の少なくとも部分的な構造表現であって、

前記構文要素の内の一要素に対し、

前記一要素のタイプを識別する識別ステップと、

前記タイプが第1タイプであれば、そのハッシュ表現を決定することによってその要素を処理する処理ステップと、

前記タイプが前記第1タイプであれば、前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、

を行う解析方法によって生成されることを特徴とする構造表現。

【請求項48】構文要素を含むマークアップ言語文書を解析する解析装置であって、

プロセッサと、(i) 前記文書と、(i i) 前記文書を解析する手続を前記プロセッサに実行させるプログラムを格納したプログラムと、格納するメモリを含み、

該プログラムは、

(i) 前記構文要素のタイプを識別するコードと、

(i i) 前記タイプが第1タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、

(i i i) 前記タイプが第1タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、

を含むことを特徴とする解析装置。

【請求項49】構文要素を含むVRDに照らして構文要素を含むマークアップ言語文書の妥当性を検査する検査装置であって、

(a) プロセッサと、

(b) (i) 前記文書と、(i i) 前記VRDと、(i i i) 文書の妥当性を検査する手順を前記プロセッサに実行させるプログラムと、を格納するメモリを有し、

(c) 前記プログラムは、

(c a) マークアップ言語文書を、そこで識別されるド

キュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第1ドキュメントタグでなければ、処理するコードを有し、該コードは、

(c a a) 前記ドキュメントタグの階層位置を決定するコードと、

(c a b) 前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、

(c a c) 前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、

前記プログラムは更に、

(c b) V R Dを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理するコードを有し、該コードは、

(c b a) 前記タグの階層位置を決定するコードと、

(c b b) 対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、

(c b c) 前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、

前記プログラムは更に、

(c c) 前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの適当なサブセットであるかのいずれかであることを立証し、それにより前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とする検査装置。

【請求項50】 構文要素を含むV R Dに照らして構文要素を含むマークアップ言語文書の妥当性を検査する検査装置であって、

(a) プロセッサと、

(b) (i) 前記文書と、(i i) 前記V R Dと、(i i i) 文書の妥当性を検査する手順を前記プロセッサに実行させるプログラムと、を格納するためのメモリを有し、

(c) 前記プログラムは、

(c a) 前記V R Dを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、

(c a a) 前記構造要素の構文属性を決定するコードと、

(c a b) 前記構造要素のハッシュ表現を決定するコードと、

(c a c) 前記V R Dの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、

前記プログラムは更に、

(c b) マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含

み、該コードは、

(c b a) 前記文書構造要素の構文属性を決定するコードと、

(c b b) 前記文書構造要素のハッシュ表現を決定するコードと、

(c b c) 前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、

前記プログラムは更に、

(c c) 前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現を、前記V R Dの前記構造表現における対応構文属性及びハッシュ表現と比較し、それにより前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とする検査装置。

【請求項51】 V R Dに照らしてマークアップ言語文書の妥当性を検査を行う検査方法であって、

前記V R Dにおける第1タイプの最も深くネストされた構文要素についての第1拡張ハッシュ表現を決定するステップと、

V R Dリストに前記第1拡張ハッシュ表現を格納するステップと、

マークアップ言語文書における第1タイプの最も深くネストされた構文要素について、第2拡張ハッシュ表現を決定するステップと、

前記第2拡張ハッシュ表現が前記V R Dリストに存在する場合に、前記マークアップ言語文書は妥当でないことを宣言するステップと、

を含むことを特徴とする検査方法。

【請求項52】 前記第1タイプの前記構文要素は、構造要素及びその一部、

前記構造要素の定義、

前記構造要素の宣言、

前記構造要素のマッチ、

のいずれかであることを特徴とする請求項51に記載の検査方法。

【請求項53】 前記構造要素はタグであることを特徴とする請求項52に記載の検査方法。

【請求項54】 前記V R Dの前記最も深くネストされた構文要素は、前記V R Dの広域的構造、或は前記V R Dの局所的サブ構造のいずれかの中に最も深くネストされた構文要素であり、

前記マークアップ言語文書の最も深くネストされた構文要素は、前記マークアップ言語文書の広域的構造、或は、前記マークアップ言語文書の局所的サブ構造のいずれかの中に最も深くネストされた構文要素であることを特徴とする請求項51に記載の検査方法。

【請求項55】 V R Dに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、

前記V R Dの第1タイプの最も深くネストされた構文要素について第1拡張ハッシュ表現を決定する手段と、

VRDリストの前記第1拡張ハッシュ表現を格納する手段と、

前記マークアップ言語文書における第1タイプの最も深くネストされた構文要素について、第2拡張ハッシュ表現を決定する手段と、

前記第2拡張ハッシュ表現が前記VRDリストに存在する場合に、前記マークアップ言語文書は妥当でないと宣言する手段と、

を有することを特徴とする検査装置。

【請求項56】VRDに照らしてマークアップ言語文書の妥当性を検査する手続をコンピュータに実行させるコンピュータプログラムであって、

前記VRDの第1タイプの最も深くネストされた構文要素について第1拡張ハッシュ表現を決定するコードと、VRDリストの前記第1拡張ハッシュ表現を格納するコードと、

前記マークアップ言語文書における第1タイプの最も深くネストされた構文要素について、第2拡張ハッシュ表現を決定するコードと、

前記第2拡張ハッシュ表現が前記VRDリストに存在する場合に、前記マークアップ言語文書は妥当でないと宣言するコードと、

を含むことを特徴とするコンピュータプログラム。

【請求項57】VRDに照らしてマークアップ言語文書の妥当性を検査する手続をコンピュータに実行させるコンピュータプログラムを格納したコンピュータに含まれるコンピュータプログラム製品であって、該プログラムは、

前記VRDの第1タイプの最も深くネストされた構文要素について第1拡張ハッシュ表現を決定するコードと、VRDリストに前記第1拡張ハッシュ表現を格納するコードと、

前記マークアップ言語文書における前記第1タイプの最も深くネストされた構文要素について第2拡張ハッシュ表現を決定するコードと、

前記第2拡張ハッシュ表現が前記VRDリストに存在する場合に、前記マークアップ言語文書が妥当でないと宣言するコードと、

を含むことを特徴とするコンピュータプログラム製品。

【請求項58】前記コードは、英数字を含むことを特徴とする請求項7に記載の解析方法。

【発明の詳細な説明】

【0001】

【著作権についての注意】この特許明細書は、著作権保護の対象となる素材を含む。その著作権者は、特許庁のファイルから閲覧する目的で、本特許明細書、或は、関連素材を複製することに対しては、なんらの異論もないが、そうでなければ、全てのいかなる著作権もこれを保有する。

【0002】

【発明の属する技術分野】本発明は、一般的にマルチメディア文書の処理、特に、マークアップ言語で記述された文書に関する。本発明は、マークアップ言語文書を構文解析する方法及び装置に関するものである。本発明はまた、マークアップ言語で記述された文書を構文解析するための処理をコンピュータに実行させるために作成されたコンピュータプログラムに関し、更に、このコンピュータプログラムが記録されたコンピュータ可読媒体を含むコンピュータプログラム製品に関する。

【0003】

【技術的背景】構文解析（パーシング）とは、文書から情報を抽出する処理である。この処理には、通常、文書の構文に対する最低限のチェックが少なくとも含まれる。そして、一般的に、その文書のツリー構造での記述、又は、イベントの論理的なつながり（logical chain）を得ることができる。そのイベントの論理的なつながりに基づいた構造表現は、一般的に、文頭から文末まで、文書を順番に構文解析することによって生成される。

【0004】ツリー型のパーサは、例えば、XML文書を内部ツリー構造にコンパイルし、アプリケーションが扱うことのできる階層モデルを与える。W3Cの文書オブジェクトモデル（DOM）作業グループは、拡張可能なマークアップ言語（XML）文書に用いる標準ツリー型アプリケーションプログラムインターフェース（API）を現在開発している。他方、イベント型のパーサは各要素の開始や終了等の解析イベントを、アプリケーション—このアプリケーションのために解析が行われている—に直接報告する。この報告は典型的にはコールバックを用いて実行され、内部ツリー構造を必要としない。解析が必要なアプリケーションは、様々なイベントを処理するためにハンドラを実行する。これはグラフィカルユーザインターフェースにおけるイベントのハンドリングによく似ている。

【0005】ツリー型のパーサは、広範囲のアプリケーションに有用だが、一般的に、システムリソースに負担がかかり、特に、解析対象となっている文書が大きな場合には負担となる。さらに、アプリケーションは、そのアプリケーション独自のツリー構造の構築を必要とする場合があるが、ツリー表現を、異なる表現と照らし合わせるためだけに、構築するのは非効率である。イベント型のパーサによれば、XML文書に対し、より単純で、より低いレベルでのアクセスが可能となり、使用可能なシステムメモリよりも大きなメモリを要する文書の解析を容易にする。“Simple API for XML”（SAXパーサと称す）は、XML文書を解析するためのイベント駆動型インターフェースである。SAXパーサは、図2A、図2B、図3A、図3B、図3Cにおいて、更に詳細に議論する。

【0006】図1A、図1Bは、パーサシステムのプロ

ック図を示したものである。ここでは、以下のXML文書部分106について考察する。

```

105 <:Shakespeare>;
110 <:|—This is a comment—>;
115 <:div class="preface"Name1="value1"name2="value2">;
120 <:mult list=&:lt:><:/mult>;
125 <:banquo>;
130 Say
135 <:quote>;
140 goodnight<:/quote>;
145 Hamlet.<:/banquo>;
150 <:Hamlet><:quote>:Goodnight, Hamlet.<:/quote><:/Hamlet>;
155 <:/Shakespeare>;

```

【0007】

[1]

【0008】図1Bにおいて、XML文書106は、この例では、イベント型のパーサである、パーサ112へ入力される。点線枠108で示されるように、文書型定義(DTD)又はXMLスキーマもまた、パーサ112へオプションとして入力される。パーサ112は、矢印114で示されるように、簡単なリストから成る文書106の部分的な構造表現を出力する。図1Aにおいて、カスケードスタイルシート(CSS)又は拡張スタイルシート(XSL)104は、CSS又はXSLパーサ110へ入力される。DTD102もまたこのパーサ110へ入力される。本図において、XMLパーサ112及びCSS/XSLパーサ110は共に、イベント駆動型パーサである。

【0009】XMLのようなマークアップ言語の利点の1つは、タグを使用して文書の様々な部分を定義することにより、文書を、容易に、よりスマートで、よりポータブルで、よりパワフルにできることにある。この機能は、XMLの記述上の性質からきている。XML文書はサブジェクト単位でカスタマイズされる。そして、それにより、カスタマイズされたタグは、人間が文書を構造的に理解できるようにするために用いられる。しかし、この特性によって、XML文書が冗長で大きなものになることがよくあり、場合によっては問題となる。例えば、XML文書がプリンタのように、ハードウェア制約のある装置で解析されなければならない場合、メモリをたくさん使用する性質のある従来型の解析は、そのような装置に格納できる限られたメモリと競合してしまう。さらに、ハードウェア制約のある装置によって文書が処理される場合には、人間のXML文書の読みやすさは、一般的に、余り利点にはならない。さらに、XML文書の解析において、タグ文字列のマッチング操作が多く必要とされ、許容できないほどの処理要求の負担がかかり、ひいては、許容できない数のプロセッササイクルを引き起す。これらの問題は、図1A及び図1Bに示される両方のパーサ例に当てはまる。

【0010】

【本発明の開示】本発明の目的は、既存の機器の一つ以

上の欠点をほぼ無くし、或は、少なくとも改善することにある。

【0011】本発明の第1の態様では、構文要素を含むマークアップ言語文書を解析する解析方法であって、前記構文要素の内の一要素に対し、前記一要素のタイプを識別する識別ステップと、前記タイプが第1タイプであれば、そのハッシュ表現を決定することによってその要素を処理する処理ステップと、前記タイプが前記第1タイプであれば、前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、を含むことを特徴とする解析方法が提供される。

【0012】本発明の他の態様では、VRDに照らしてマークアップ言語文書の妥当性検査を行う検査方法であって、(a)マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第1ドキュメントタグでなければ、処理するステップを有し、該処理は、(i)前記ドキュメントタグの階層位置を決定するステップと、(ii)前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するステップと、(iii)前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するステップと、を含み、前記検査方法は更に、

(b)VRDを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理するステップを有し、該処理は、(i)前記タグの階層位置を決定するステップと、(ii)対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するステップと、(iii)前記タグの前記拡張ハッシュ表現をリストに格納するステップと、を含み、前記検査方法は更に、(c)前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合には、前記マークアップ言語文書の妥当性を検査するステップを有することを特徴とする検査方法が提

供される。

【0013】本発明の他の態様では、VRDに照らしてマークアップ言語文書の妥当性を検査する検査方法であって、(a)前記VRDを、そのなかで識別された構造要素ごとに、処理する処理ステップを含み、該処理ステップは、(i)前記構造要素の構文属性を決定するステップと、(ii)前記構造要素のハッシュ表現を決定するステップと、(iii)前記VRDの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するステップと、を含み、前記検査方法は更に、

(b)マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理する処理ステップを含み、該処理ステップは、(i)前記文書構造要素の構文属性を決定するステップと、(ii)前記文書構造要素のハッシュ表現を決定するステップと、(iii)前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するステップと、を含み、前記検査方法は更に、(c)前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記VRDの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するステップを有することを特徴とする検査方法が提供される。

【0014】本発明の他の態様では、構文要素を含むマークアップ言語文書を符号化する符号化方法であって、前記構文要素の一つについて、構文要素のタイプを識別するステップと、(i)前記タイプが第1タイプであれば、前記構文要素のハッシュ表現を決定すること、(ii)前記タイプが第1タイプでなければ、前記構文要素の圧縮表現を決定すること、(iii)前記構文要素を保持すること、のいずれかによって、前記構文要素を処理するステップと、を含むことを特徴とする符号化方法が提供される。

【0015】本発明の他の態様では、符号化構文要素を含むマークアップ言語文書を復号する復号方法であって、前記符号化構文要素の一つについて、前記符号化構文要素のタイプを識別するステップと、(i)前記タイプが第1タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、(ii)前記タイプが第1タイプでなければ、前記符号化構文要素の解凍表現を決定すること、(iii)前記符号化構文要素を保持すること、の少なくとも一つで、前記符号化構文要素を処理するステップと、を含むことを特徴とする復号方法が提供される。

【0016】本発明の他の態様では、構文要素を含むマークアップ言語文書を解析する解析装置であって、前記構文要素のタイプを識別する識別手段と、前記タイプが第1タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理する処理手段と、前記タイプが第1タイプである場合に前記ハッシュ表現

を用いて、前記文書の少なくとも部分的構造表現を増大する増大手段と、を含むことを特徴とする解析装置が提供される。

【0017】本発明の他の態様では、VRDに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、(a)マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第1ドキュメントタグでなければ、処理する手段を有し、該手段は、(i)前記ドキュメントタグの階層位置を決定する手段と、(ii)前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定する手段と、(iii)前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納する手段と、を含み、前記検査装置は更に、(b)VRDを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理する手段を有し、該手段は、(i)前記タグの階層位置を決定する手段と、(ii)対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定する手段と、(iii)前記タグの前記拡張ハッシュ表現をリストに格納する手段と、を含み、前記検査装置は更に、(c)前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバーの妥当なサブセットであるかのどちらかであることを立証し、それによって、前記マークアップ言語文書の妥当性を検査する手段を有することを特徴とする検査装置が提供される。

【0018】本発明の他の態様では、VRDに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、(a)前記VRDを、そのなかで識別された構造要素ごとに、処理する手段を含み、該手段は、(i)前記構造要素の構文属性を決定する手段と、(ii)前記構造要素のハッシュ表現を決定する手段と、(iii)前記VRDの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納する手段と、を含み、前記検査装置は更に、(b)マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理する手段を含み、該手段は、(i)前記文書構造要素の構文属性を決定する手段と、(ii)前記文書構造要素のハッシュ表現を決定する手段と、(iii)前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納する手段と、を含み、前記検査装置は更に、(c)前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現と、前記VRDの前記構造表現における対応構文属性及びハッシュ表現とを比較し、それにより、前記マークアップ言語文書の妥当性を検査する手段を有することを特徴とする検

査装置が提供される。

【0019】本発明の他の態様では、構文要素を含むマークアップ言語文書を符号化し、前記文書の少なくとも部分的構造表現を形成する符号化装置であって、構文要素のタイプを識別する手段と、(i)前記タイプが第1タイプであれば、前記構文要素のハッシュ表現を決定すること、(ii)前記タイプが第1タイプでなければ、前記構文要素の圧縮表現を決定すること、(iii)前記構文要素を保持すること、のいずれかによって、前記構文要素を処理する手段と、を含むことを特徴とする符号化装置が提供される。

【0020】本発明の他の態様では、符号化構文要素を含むマークアップ言語文書を復号する復号装置であって、前記符号化構文要素のタイプを識別する手段と、

(i)前記タイプが第1タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、(ii)前記タイプが第1タイプでなければ、前記符号化構文要素の解凍表現を決定すること、(iii)符号化構文要素を保持すること、の少なくとも一つによって、前記符号化構文要素を処理する手段と、を含むことを特徴とする復号装置が提供される。

【0021】本発明の他の態様では、コンピュータに構文要素を含むマークアップ言語文書を解析する手順を実行させるコンピュータプログラムであって、前記構文要素のタイプを識別するコードと、前記タイプが第1タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、前記タイプが第1タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、を含むことを特徴とするコンピュータプログラムが提供される。

【0022】本発明の他の態様では、コンピュータにVRDに照らしてマークアップ言語文書の妥当性を検査する手順を実行させるコンピュータプログラムであって、

(a)マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第1ドキュメントタグでなければ、処理するコードを有し、該コードは、(i)前記ドキュメントタグの階層位置を決定するコードと、(ii)前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、(iii)前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、前記コンピュータプログラムは更に、(b)VRDを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理するコードを有し、該コードは、(i)前記タグの階層位置を決定するコードと、(ii)対応するタグ階層にお

る前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、(iii)前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、前記コンピュータプログラムは更に、(c)前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合には、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラムが提供される。

【0023】本発明の他の態様では、コンピュータにVRDに照らしてマークアップ言語文書の妥当性を検査する手順を実行させるコンピュータプログラムであって、

(a)前記VRDを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、(i)前記構造要素の構文属性を決定するコードと、(ii)前記構造要素のハッシュ表現を決定するコードと、(iii)前記VRDの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、前記プログラムは更に、(b)マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、(i)前記文書構造要素の構文属性を決定するコードと、(ii)前記文書構造要素のハッシュ表現を決定するコードと、(iii)前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、前記プログラムは更に、(c)前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記VRDの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラムが提供される。

【0024】本発明の他の態様では、コンピュータに構文要素を含むマークアップ言語文書を符号化する手順を実行させるコンピュータプログラムであって、構文要素のタイプを識別するコードと、(i)前記タイプが第1タイプであれば、前記構文要素のハッシュ表現を決定すること、(ii)前記タイプが第1タイプでなければ、前記構文要素の圧縮表現を決定すること、(iii)前記構文要素を保持すること、のいずれかによって、前記構文要素を処理するコードと、を含むことを特徴とするコンピュータプログラムが提供される。

【0025】本発明の他の態様では、コンピュータに符号化構文要素を含むマークアップ言語文書を復号する手順を実行させるコンピュータプログラムであって、前記符号化構文要素のタイプを識別するコードと、(i)前記タイプが第1タイプであれば、前記符号化構文要素の逆ハッシュ表現を決定すること、(ii)前記タイプが第1タイプでなければ、前記符号化構文要素の解凍表現を決定すること、(iii)符号化構文要素を保持する

こと、の少なくとも一つで、前記符号化構文要素を処理するコードと、を含むことを特徴とするコンピュータプログラムが提供される。

【0026】本発明の他の態様では、コンピュータに構文要素を含むマークアップ言語文書を解析する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、前記コンピュータプログラムは、前記構文要素のタイプを識別するコードと、前記タイプが第1タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、前記タイプが第1タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的構造表現を増大するコードと、を含むことを特徴とするコンピュータプログラム製品が提供される。

【0027】本発明の他の態様では、コンピュータにVRDに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、前記コンピュータプログラムは、(a)マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のドキュメントタグ階層における第1ドキュメントタグでなければ、処理するコードを有し、該コードは、(i)前記ドキュメントタグの階層位置を決定するコードと、(ii)前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、(iii)前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、前記コンピュータプログラムは更に、

(b)VRDを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理するコードを有し、該コードは、(i)前記タグの階層位置を決定するコードと、(ii)対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、(iii)前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、前記コンピュータプログラムは更に、(c)前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットである場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム製品が提供される。

【0028】本発明の他の態様では、コンピュータにVRDに照らしてマークアップ言語文書の妥当性を検査する手続を実行させるコンピュータプログラムを、格納したコンピュータ可読媒体を含むコンピュータプログラム製品であって、前記プログラムは、(a)前記VRD

を、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、(i)前記構造要素の構文属性を決定するコードと、(ii)前記構造要素のハッシュ表現を決定するコードと、(iii)前記VRDの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、前記プログラムは更に、(b)マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、(i)前記文書構造要素の構文属性を決定するコードと、(ii)前記文書構造要素のハッシュ表現を決定するコードと、(iii)前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、前記プログラムは更に、(c)前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現が、前記VRDの前記構造表現における対応構文属性及びハッシュ表現に一致する場合に、前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とするコンピュータプログラム製品が提供される。

【0029】本発明の他の態様では、構文要素を含むマークアップ言語文書の少なくとも部分的な構造表現であって、前記構文要素の内の一要素に対し、前記一要素のタイプを識別する識別ステップと、前記タイプが第1タイプであれば、そのハッシュ表現を決定することによってその要素を処理する処理ステップと、前記タイプが前記第1タイプであれば、前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大させる増大ステップと、を行う解析方法によって生成されることを特徴とする構造表現が提供される。

【0030】本発明の他の態様では、構文要素を含むマークアップ言語文書を解析する解析装置であって、プロセッサと、(i)前記文書と、(ii)前記文書を解析する手続を前記プロセッサに実行させるプログラムを格納したプログラムと、格納するメモリを含み、該プログラムは、(i)前記構文要素のタイプを識別するコードと、(ii)前記タイプが第1タイプである場合に、前記構文要素を、そのハッシュ表現を決定することによって、処理するコードと、(iii)前記タイプが第1タイプである場合に前記ハッシュ表現を用いて、前記文書の少なくとも部分的な構造表現を増大するコードと、を含むことを特徴とする解析装置が提供される。

【0031】本発明の他の態様では、構文要素を含むVRDに照らして構文要素を含むマークアップ言語文書の妥当性を検査する検査装置であって、(a)プロセッサと、(b)(i)前記文書と、(ii)前記VRDと、(iii)文書の妥当性を検査する手順を前記プロセッサに実行させるプログラムと、を格納するメモリを有し、(c)前記プログラムは、(ca)マークアップ言語文書を、そこで識別されるドキュメントタグごとに、該ドキュメントタグが、対応するマークアップ言語のド

キュメントタグ階層における第1ドキュメントタグでなければ、処理するコードを有し、該コードは、(c a a) 前記ドキュメントタグの階層位置を決定するコードと、(c a b) 前記ドキュメントタグ階層における前のドキュメントタグのハッシュ表現に連結された、前記ドキュメントタグの拡張ハッシュ表現を決定するコードと、(c a c) 前記ドキュメントタグが前のドキュメントタグよりも深くネストにされる場合に、前記ドキュメントタグの前記拡張ハッシュ表現を格納するコードと、を含み、前記プログラムは更に、(c b) VRDを、そこで識別されるタグごとに、該タグが対応タグ階層の第1タグでない場合に、処理するコードを有し、該コードは、(c b a) 前記タグの階層位置を決定するコードと、(c b b) 対応するタグ階層における前のタグのハッシュ表現に連結される前記タグの拡張ハッシュ表現を決定するコードと、(c b c) 前記タグの前記拡張ハッシュ表現をリストに格納するコードと、を含み、前記プログラムは更に、(c c) 前記ドキュメントタグの前記拡張ハッシュ表現が、前記リスト内にあるか、或は、前記リストのメンバの妥当なサブセットであるかのいずれかであることを立証し、それにより前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とする検査装置が提供される。

【0032】本発明の他の態様では、構文要素を含むVRDに照らして構文要素を含むマークアップ言語文書の妥当性を検査する検査装置であって、(a) プロセッサと、(b) (i) 前記文書と、(i i) 前記VRDと、(i i i) 文書の妥当性を検査する手順を前記プロセッサに実行させるプログラムと、を格納するためのメモリを有し、(c) 前記プログラムは、(c a) 前記VRDを、そのなかで識別された構造要素ごとに、処理するコードを含み、該コードは、(c a a) 前記構造要素の構文属性を決定するコードと、(c a b) 前記構造要素のハッシュ表現を決定するコードと、(c a c) 前記VRDの構造表現における、前記構造要素の前記ハッシュ表現及び構文属性を格納するコードと、を含み、前記プログラムは更に、(c b) マークアップ言語文書を、そのなかで識別されたドキュメント構造要素ごとに、処理するコードを含み、該コードは、(c b a) 前記文書構造要素の構文属性を決定するコードと、(c b b) 前記文書構造要素のハッシュ表現を決定するコードと、(c b c) 前記文書の構造表現における、前記文書構造要素の前記ハッシュ表現と構文属性を格納するコードと、を含み、前記プログラムは更に、(c c) 前記文書の前記構造表現における前記各文書構造要素の構文属性及びハッシュ表現を、前記VRDの前記構造表現における対応構文属性及びハッシュ表現と比較し、それにより前記マークアップ言語文書の妥当性を検査するコードを含むことを特徴とする検査装置が提供される。

【0033】本発明の他の態様では、VRDに照らして

マークアップ言語文書の妥当性検査を行う検査方法であって、前記VRDにおける第1タイプの最も深くネストされた構文要素についての第1拡張ハッシュ表現を決定するステップと、VRDリストに前記第1拡張ハッシュ表現を格納するステップと、マークアップ言語文書における第1タイプの最も深くネストされた構文要素について、第2拡張ハッシュ表現を決定するステップと、前記第2拡張ハッシュ表現が前記VRDリストに存在する場合に、前記マークアップ言語文書は妥当でないことを宣言するステップと、を含むことを特徴とする検査方法が提供される。

【0034】本発明の他の態様では、VRDに照らしてマークアップ言語文書の妥当性を検査する検査装置であって、前記VRDの第1タイプの最も深くネストされた構文要素について第1拡張ハッシュ表現を決定する手段と、VRDリストの前記第1拡張ハッシュ表現を格納する手段と、前記マークアップ言語文書における第1タイプの最も深くネストされた構文要素について、第2拡張ハッシュ表現を決定する手段と、前記第2拡張ハッシュ表現が前記VRDリストに存在する場合に、前記マークアップ言語文書は妥当でないことを宣言する手段と、を有することを特徴とする検査装置が提供される。

【0035】本発明の他の態様では、VRDに照らしてマークアップ言語文書の妥当性を検査する手順をコンピュータに実行させるコンピュータプログラムであって、前記VRDの第1タイプの最も深くネストされた構文要素について第1拡張ハッシュ表現を決定するコードと、VRDリストの前記第1拡張ハッシュ表現を格納するコードと、前記マークアップ言語文書における第1タイプの最も深くネストされた構文要素について、第2拡張ハッシュ表現を決定するコードと、前記第2拡張ハッシュ表現が前記VRDリストに存在する場合に、前記マークアップ言語文書は妥当でないことを宣言するコードと、を含むことを特徴とするコンピュータプログラムが提供される。

【0036】本発明の他の態様では、VRDに照らしてマークアップ言語文書の妥当性を検査する手順をコンピュータに実行させるコンピュータプログラムを格納したコンピュータに含まれるコンピュータプログラム製品であって、該プログラムは、前記VRDの第1タイプの最も深くネストされた構文要素について第1拡張ハッシュ表現を決定するコードと、VRDリストに前記第1拡張ハッシュ表現を格納するコードと、前記マークアップ言語文書における前記第1タイプの最も深くネストされた構文要素について第2拡張ハッシュ表現を決定するコードと、前記第2拡張ハッシュ表現が前記VRDリストに存在する場合に、前記マークアップ言語文書が妥当でないことを宣言するコードと、を含むことを特徴とするコンピュータプログラム製品が提供される。

【0037】

【ベストモードを含む詳細な説明】以下、添付の図面を参照して説明するが、ここでは、同じ符号を付したステップ及び／又は特徴は、逆の意図が表されない限り、同じ機能或は動作を有するものとする。

【0038】本明細書に記載の発明の概念は、XMLタグについて、及び、可能ならばXMLファイル中の他の要素について、“完全な”ハッシュを実行することによって、XMLパーサのメモリ必要量を減少させ、様々な性能値を向上させることができる、という考えに基づいている。ハッシュ関数とは、数学的であってもなくてもよいが、入力文字列を取得し、ハッシュ値と呼ばれる出力コード番号に変換する関数である。完全なハッシュ関数は、予めセットされた領域内のユニークな入力文字列に対して、ユニークなコード番号を作る関数である。入力文字列は、例えば、W3Cで許可された英数字や他の文字で構成され、ハッシュ処理の仕様に記されている特定の長さよりも短くなければならない。それに代えて、又は、それに加えて、入力文字列は、例えば、入力内容に基づいたコード番号の衝突が起こる可能性があるという点から、別の方法で制約を加えることができる。このアイデアによって、任意のXMLタグを、数やコードとして扱うことができる。そして、メモリ内に数字の形で格納することができる。パーサは、通常、構造が解析されるときに、メモリ内のXML構造の一部分を保存するので、XMLタグをユニークな数字に変換することによってメモリ必要量を減少させることができ、更に、文字列と文字列の比較を、同様の、しかしより高速の数値の比較に置き換えることができる。

【0039】ここで記載する装置の原理は、非常に様々なマークアップ言語を用いた文書の解析にも、普通に適用することができる。説明を簡潔にするため、ここで開示される装置は、XML言語を参照して、説明することとする。しかし、これは発明の技術的範囲を制限しようとするものではない。例えば、ここに記載された機構はまた、UTF-16転送フォーマットに適用することができる（UTF-16の詳細は、国際規格ISO/IEC 10646-1を参照のこと）。

【0040】図2A及び図2Bは、従来技術のSAXパーサ処理236を示したものである。これは、随意、整形性、及び／又は、妥当性をチェックするサブ処理をサポートする。

【0041】図2Aにおいて、マークアップ文書（本実施形態においては、XMLドキュメント）は、初期ステップ200で開かれる。その後、決定ステップ202は、文書が未処理の（すなわち解析されていない）文字を含んでいるかどうかをテストし、決定ステップ202の条件に合ったなら、以下のステップ204で文字を読み込み、文字列に格納する。しかし、テストステップ202において、更に文字が検出されなければ、解析処理236はステップ234で終了する。

【0042】ステップ204に続いて、テストステップ206において、チェックが行われ、完全な構文要素が組み立てられているかどうかを判断する。もし完全な構文要素が組み立てられているのであれば、パーサ処理236は“構文タイプ”識別ステップ210へ進む。反対に、完全な構文要素が組み立てられていなければ、パーサ処理236は文書中に使用できる文字が更に存在するかを判断する判断ブロック208へ向かう。さらに文字の使用が可能であれば、パーサ処理236は“はい”矢印に応じてステップ204へ戻る。一方、これ以上文字が使用できない場合は、処理236は“いいえ”矢印に応じて、構文要素タイプ識別ステップ210へ向かう。

【0043】“タイプ識別”ステップ210は、組み立てられた構文要素に対して“タイプ”を識別し、その後、ステップ212で、要素文字列が文書構造のメモリ表現の中に置かれ、それによりこの時点で組み立てられていたように表現を増やす。その文章構造のメモリ表現は、一般的に、イベント駆動型パーサの場合、文書の部分構造表現であり、単純なリストとなり得る。

【0044】ステップ212の後、処理236は整形性チェックを実行すべきかどうかを判断するテストステップ242へ向かう。整形性チェックは、文章が、インターネット上の <http://www.w3.org/tr/2000/rec-XML-20001006.html> で入手できる『拡張可能なマークアップ言語（XML）1.0（第2版）W3C勧告、2000年10月6日』の5ページに定義されているような、適切な“整形性制約”を満たすことをチェックする。整形性チェックは、文章が一般的な構造ルールに準拠しているかどうかをテストする、特に、文書中のタグが適切にネストになっているかどうかをテストする。そのようなチェックが実行されると、処理236は“YES”矢印に応じて、点線で示された境界線246上の“a”に向かう。参照文字“a”から“d”に沿った点線246は、図2Bの対応する境界線に映されており、そこで処理236がさらに示されている。整形性チェックが実行されなかった場合、処理236は“いいえ”矢印に応じて、テストステップ242から“妥当性チェック”を実行すべきかどうかを判断するテストステップ244へ向かう。妥当性チェックは、前記W3C勧告5.1節に示される文書型定義（DTD）のような、妥当性参照文書（簡単のためVRDと称す）に定義される妥当性の制約に照らして、文書中の構文要素の比較を伴う。DTDとXMLスキーマは妥当性チェックが実行されるVRDの例であるが、ここで述べられている妥当性チェックは他のタイプのVRDに対して実行される。この比較処理は、単なる整形性チェックに比し、より広範囲に渡って、要素の正しい構文配置を検証する。

【0045】妥当性チェックが実行されると、処理236は“YES”矢印に応じて、点線で示された境界線246上の“b”に向かう。他方、妥当性チェックが実行

されなかった場合、処理236は”NO”矢印に応じ、点線で示された境界線246上の”c”に向かう。整形性チェックが選択された場合、処理236は境界線246上の”a”からオプションサブ処理238、特にその中の整形性チェックステップ214へ向かう。処理238のオプションとしての性質は、点線で囲まれた長方形で示される。

【0046】妥当性チェックが選択された場合、処理236は境界線246上の”b”からオプションサブ処理240、特にその中に見られる妥当性チェックステップ220へ向かう。処理240の選択的な性質は点線で囲まれた長方形で示される。妥当性チェックが選択されない場合、処理236は境界線246上の”c”からアクション選択ステップ226へ向かう。整形性チェックが選択されて、整形性ステップ214の後、次の誤りチェックステップ216で誤りが検出された場合、矢印218で示されるように修正動作及び／または誤り指示が行われる。他方、誤りが検出されなければ、パーサ処理236はステップ216からサブ処理240へ向かい、ここでは妥当性チェックがステップ220で実行される。前述のように、解析処理236は、誤りチェックステップ216から、もしくは整形性チェックサブ処理238をバイパスして、妥当性チェックステップ220に向かい、処理236は境界線246上の”b”から妥当性チェックステップ220に直接向かう。オプションの整形性サブ処理238は、適切な判断がテストステップ242、244で下された場合にバイパスすることができる(図2A参照)。

【0047】前述のように、妥当性チェックステップ220は、文書型定義(DTD)に対して考慮されるマークアップ文書中の識別された構文要素の比較を含む。この比較処理は、サブ処理238に示されている単なる整形性チェックよりも広範囲に渡って、要素の正しい構文配置を検証する。

【0048】妥当性チェックステップ220に続いて、誤りチェックステップ222で誤りが検出されると、修正動作が実行され、矢印224で示されるようにはい／いいえ誤り指示が実行される。一方、誤りが検出されない場合は、パーサ処理236はアクション選択処理226へ進み、そこでは考慮されている構文要素のタイプに基づいてアクションが選択される。図2Aのステップ242及び244の決定において適切な判断が下されると、選択的サブ処理238及び240の両方をバイパスすることができる。両方の上記サブ処理がバイパスされると、前述のようにパーサ処理236は境界線246上の”c”から直接アクション選択ステップ226へ向かう。

【0049】構文要素がタグであった場合、矢印228で示されるように、タグ値や対応する文字列は、解析処理が実行されているアプリケーションに送られ、タグの

メモリ表現が保存される。他方、要素タイプがタグのタイプでない場合、矢印230で示されるように、要素値文字列は関連アプリケーションに送られ、要素のメモリ表現は削除される。最終的に、パーサ処理236は、矢印232で示されるように、図2Bの点線で示された境界線上の”d”へ向かい、図2Aの対応する点線で示された境界線246上の”d”から文字テストステップ202へ向かう。

【0050】XML文書の冗長性により、多くのメモリ必要量が生じ、それに対応して、元の文字列形式中の文書構造を格納するために大量のメモリ必要量を要することになる。この文書構造はステップ212において参照される。さらに、可変長の英数字文字列間での文字列比較のパフォーマンスに関し、かなりの処理負荷が整形性チェックステップ214及び妥当性チェックステップ220の両方で生じる。

【0051】次に示す2つのステップの両方に関連して、一般的に、文書の部分メモリ表現は格納されなければならない。つまり、(i)階層分岐が閉じているかどうかのチェック、すなわち終了タグと開始タグのマッチング、また、分岐の重複がないかどうかのチェックについてのステップ214と、(ii)DTDに対する構造の一致性やタグ名のチェックに加えて、(i)と同様の処理が必要となるステップ220と、である。パーサは、通常、XML構造が解析される場合、メモリ中にXML構造の一部分を保存しなければならない。SAXパーサの場合でも、正しく動作するために、XML構造の局所的な部分がメモリ中に格納されなければならない。しかし、各XMLタグがハッシュ関数を用いてユニークな数字に変換される場合、ハッシュ演算で生じる数字は、関連する任意の長さのXMLタグ文字列よりも小さいので、メモリ必要量は一般的に減少する。さらに、開始&終了タグのマッチングに必要な文字列間の比較は、より高速な数値の比較で置き換えることができるため、処理負荷が減少する。

【0052】代表的ハッシュアルゴリズムは、(i)巡回冗長符号化アルゴリズム(CRC)(通例、データ転送や格納において、署名分析又は誤り検出／訂正に用いられる)と、(ii)完全ロスレス符号化アルゴリズムと、(iii)ホフマン符号化アルゴリズムと、を含む。

【0053】一般的に、適切なハッシュアルゴリズムは、静的な演算でなければならない。言い換えれば、要求されたデータセットに対し、同一の入力条件においては常に同じハッシュ結果を出力しなければならない。しかし、要求されたデータセットはその環境によって変化する。そして、そのようなデータセットは、通常、少なくとも一つの全マークアップ文書を含むことができ、また、関連DTD或いはXMLスキーマや、リンクされ

たマークアップ文書や、XML文書で参照されたCSS文書のような、異なった言語における関連の又はリンクされたマークアップ文書を含むこともできる。一方、タグ構文に遭遇した場合には、例えば、XML文書に非リテラル'<'文字を見つけた場合には、常に、そのアルゴリズムをリセットすることによって、必要に応じて静的ハッシュアルゴリズムを使用することができる。また、XML DTD文書に<!ELEMENT文字列を見つけた場合に、又は、CSS文書中であって有効なタグセレクトが許される全ての場合に、ハッシュアルゴリズムをリセットすることができる。

【0054】入力されたマークアップ文書中のリファレンスは、適切なハッシュアルゴリズムに信号を送るか又はそれを選択するために用いられる。これは他のマークアップ文書や、DTDや、スタイルシートや、文字エンコーディングや、名前空間などを参照できるのと同じように実行されうる。例えば、特定のハッシュアルゴリズムは特定の名前空間と同じものとして扱うことができる。これにより、文書中の名前空間のリファレンスを介したハッシュアルゴリズムへの間接的なリファレンスが可能となる。ハッシュアルゴリズムの実行は、関連するパラメータ化と共に、全体的に又は部分的にマークアップ言語に含まれる。このような、ハッシュアルゴリズムを参照したり含めたりする方法は、最適化に有効である。つまり、特定のマークアップ文書に関して様々なハッシュ方法が最適化される場合である。それにより、宛先装置やシステム中の性能とメモリ使用量が改善される。一方、前述の参照方法がマッチングに有用な場合がある。それは1つ以上のマークアップ文書を伴うアプリケーションであり、そのアプリケーションで、誤りのチェック又は解析又は他の機能の完了が必要で、1つ以上の他の文書（例えば、DTD）が既に同じアルゴリズムでハッシュ化されている場合である。

【0055】上記アプローチでは、更なる改善が可能で

```

205 Shakespeare
215     div
220         mult
221     /mult
225     banquo
235         quote
240         quote
245     /banquo
250     Hamlet
251         quote
252     /quote
253     /Hamlet
255 /Shakespeare

```

【0059】逆に、図3Aに示されるような、パース処理344において、タグ要素と非タグ要素とに対して異なる処理が行われた結果、ステップ318で同等な階層

あり、例えば、DTDハッシュを任意的に含むことが挙げられる。これは、（より低速な）文字列の比較に代えて、数値の比較を可能とすることにより、DTDを格納するための読み出し専用メモリ（ROM）の必要量を減少させ、XML文書の妥当性処理を高速化することができる。

【0056】図3A、3B及び3Cは、改良されたSAXパーサ処理344のアレンジの1つを示したものである。図3Aにおいて、ステップ300からステップ310は、図2Aに関して説明してきたステップ200からステップ210に対応するものである。ステップ310の後、テストステップ312において、組み立てられた構文要素の性質がタグであるか、別の要素タイプであるか確かめるためにテストされる。要素がタグの場合は、パース処理344は矢印316に従って、ハッシュステップ318へ向かう。ハッシュステップ318は、それぞれ図5及び6のプロセッサ414及び505を用いて、構文要素の単一数値表現を決定する。これによって、要素の表現を効率良くメモリ上に実現され、また、英数字領域よりも、単純で高速な数値での比較演算ができる。構文要素を示す要素文字列とそのハッシュ値の両方が、処理344の時点で保持されるが、ステップ318で、各々のメモリ418及び506（図5及び6を参照）を用いて文書構造のメモリ表現へ挿入されるのは、文字列の値ではなくハッシュ値である。ここまではこれのほうがいい。

【0057】図3A、図3B及び図3Cで示されるようなパース処理344を、より正確に理解するため、まず、図2で示したパース処理236との関係で、XML部の例[1]を解析する場合について説明する。この場合、サブ処理212において、XML部[1]から、以下のような解析されたマークアップタグの階層表現が生成される。

【0058】

[2]

表現が生成される。その同等階層表現は[3]に示される。[3]の階層表現は、解析されハッシュにされたマークアップタグから成る。これを例示するために、タグ

名の領域は、以下の表1に示されるように制約され、ハッシュマッピング（機能的にはハッシュ“関数”の適用と同等である）は次の表で示される。

【0060】

【表1】

タグ	ハッシュコード番号
Shakespeare	133
Div	326
Mult	371
Banquo	787
Quote	629
Hamlet	411

【0061】上記ハッシュマッピングに基づき、[1]で示されたXML文書は、以下のような階層表現とな

```

205 133
215      326
222      371
223      /371
225      787
235          629
240          /629
245      /787
254      411
255          629
256          /629
257      /411
255 /133

```

る。

【0062】

[3]

【0063】図3Aに戻ると、パース処理344は、ステップ314から点線で示された境界線356上の“a”へ向かう。参照文字“a”及び“b”に沿った点線356は、図3Bの対応する境界線に映されており、それに関連して処理344が更に示されている。

【0064】図3Bに移り、処理344は、点線で示された境界線356上の“a”から、整形性チェックが実行されるかどうかを判断するテストステップ350へ続く。そのチェックが実行されると、処理344は「YES」矢印に応じて、境界線358上の“c”へ向かう。参照文字“c”及び“f”に沿って点線で示された境界線358は、図3Cの対応する境界線に映されており、それに関連して処理344が更に示されている。整形性チェックが実行されなかった場合、処理344は、「NO」矢印に応じて、妥当性チェックが実行されるかどうかを判断するテストステップ352へ向かう。妥当性チェックが実行されると、処理344は「YES」矢印に応じて、境界線58上の“e”へ向かう。一方、妥当性チェックが実行されなかった場合は、処理344は境界線358上の“d”へ向かう。

【0065】図3Cに向かうと、整形性チェックが実行された場合、処理344は、点線で示された境界線358上の“c”から、整形性チェックステップ320へ向かう。一方、整形性チェックが選択されなかった場合、処理344は、点線で示された境界線358上の“e”から、妥当性チェックステップ326へ進む。整形性チ

ェックと妥当性チェックのどちらも選択されなかった場合、処理344は、点線で示された境界線358上の“d”から、アクション選択ステップ334へ進む。

【0066】整形性チェックステップ320は、対応するプロセッサ414及び505を用いて、整形性チェックを実行し、選択的処理346の一部分を形成している。処理346の選択的性質は点線を用いて示される。同様に、妥当性ステップ326はサブ処理348の一部分を形成し、その選択的性質は点線で示されている。

【0067】[3]に表される階層表現により、文字列の比較がより高速でより効率的な数値の比較に置き換えられることは明らかである。それによって関連する計算の負荷が軽減される。さらに、[3]で示される階層表現は、[1]で示される階層表現よりもメモリ効率に優れており、それに対応して[3]で示される表現は、前述したようなメモリ制約のあるアプリケーションにより適している。

【0068】図3Cに戻り、整形性チェックが選択され、ステップ320で整形性チェックが実行された後、パース処理344は誤りチェックステップ322へ向かう。ここでは、誤りが検出されると、矢印324で示されるように、修正動作が実行され、及び/又は、誤りが指示される。整形性チェックは、文書中のタグが適切にネストになっているかどうかを一般的に考慮する。従って、例えば、[2]を参照すると、“Hamlet”と“/Hamlet”のタグペアは“Shakespeare”と“/Shakespeare”

のタグペア内に適切にネストになっている。なぜなら、“Hamlet”タグペアは、完全に“Shakespeare”タグペア内にネストになっており、そのタグペア同士は、例えば、互いに重なり合っていないからである。

【0069】一方、誤りが検出されなかった場合、パース処理344はオプションの処理348へ向かう。ここにおいて、DTD及びXMLスキーマに関して、対応するプロセッサ414及び505を用いた、妥当性チェックステップ326が実行される。前述のように、妥当性チェックは整形性チェックよりも詳細にチェックを行うものである。従って、例えば、整形性チェックは、“Hamlet”のタグのペアが“Shakespeare”のタグペアの間に正しくネストになっているとみなすのに対し、妥当性チェックは、これとは異なり、“Hamlet”のタグペアが“Shakespeare”のタグペアの間に完全にネストになっているという正しいネストのチェックだけでなく、“Hamlet”のタグペアがこの方法で、正当なネストになっているかどうかのチェックも行う。例えば、実際、“Shakespeare”のタグペアが“Hamlet”のタグのペアの間にネストにならなければならない、この逆のネストではだめな場合もある。従って、妥当性チェック処理は、この場合、ネストが適切であるかどうか、すなわち完全に実行されるかどうかに加え、“Hamlet”タグペアが“Shakespeare”のタグペアの間にネストになり得るかどうか、というタグの階層表現もチェックする。

【0070】妥当性ステップ326を実行するため、DTD/XMLスキーマのメモリ表現を、ハッシュステップ318で生成されたマークアップ文書のハッシュの性質に一致するように、DTD又はXMLスキーマタグは、ハッシュステップ328で最初にハッシュにしておく。妥当性チェックステップ326は、ステップ318で生成された文書の構造表現と、ステップ328で生成されたDTD/XMLスキーマの構造表現とを比較する。そして、マークアップ文書の構文要素の正しい構文

```
505 <:133>:
110 <:!--This is a comment-->:
515 <:326 class="preface"Name1="value1"name2="value2">:
520 <:371 list=&:lt:>: <:/371>:
525 <:787>:
130 Say
535 <:629>:
540 goodnight<:/629>:,
545 Hamlet. <:/787>:
550 <:411>:<:629>:Goodnight, Hamlet.<:/629>:<:/411>:
555 <:/133>:
```

【0074】終了タグの表現は（一般的には、開始タグに<:section>:という書式を用いるのに対し、<:/section>:という書式を用いる）、様々な方法で定義することができ、それによってXML標準仕様との互換性が高くなり、低くなったりする。開始タグと終了タグは、現在

配置を検証する。なお、図2のステップ220で用いられるように、この比較に必要な文字列の比較は、ステップ328及び318のハッシュ演算の結果、ここで、図3Cにおいて、より高速でより効率的な数値の比較に置き換えられる。

【0071】妥当性チェックの後、処理344は誤りチェックステップ330へ向かう。ここでは、矢印332で示されるように、修正動作及び/または誤り指示が実行される。誤りがなかった場合、パース処理344は、アクションステップ334へ進む。ここにおいて、構文要素のタイプがタグの場合、パース処理が実行されているアプリケーションに、対応するタグの文字列が送られ、タグの文字列自体は、図5及び6にある418又は512のメモリから消去される。しかし、関連するハッシュで表されたタグのメモリ表現は保持される。従って、現在解析されているタグの文字列のコピーを除けば、タグの文字型のメモリ表現は保持されない。タグのメモリ表現は、これによりハッシュの形式だけとなる。要素の構文タイプが非タグタイプの場合、又は非タグ名タイプの場合、矢印338で示されるように、要素の値又は文字列の表現は、関連のアプリケーションへ送られ、関連のメモリ表現は消去される。パース処理344は今、矢印340で示されるように、点線で示された境界線358上の“f”へと折り返し、その後図3Bの点線で示された境界線358上の対応する“f”へ向かい、その後点線で示された境界線358上の“b”へ向かい、その後図3Aの点線で示された境界線356上の対応する“b”へ向かい、最後に文字テストステップ302へと向かう。これ以上文字が検出されなければ、パース処理344はステップ342で終了する。

【0072】XML文書部[1]は、ハッシュ形式のタグを持つようになり、以下のような形式で示される。

【0073】

[4]

の記述において、“同等のタイプ”だとみなされる点に注意しなければならない。さらに、開始タグと終了タグは一まとめにして機能する、すなわち文書内容の部分を区切ることは、2つのタグに関連性があることを意味する。さらに、開始タグと終了タグの前記書式は、終了タ

グが開始タグを修飾していることを意味し、識別文字、すなわち“/”といったものは、対応する終了タグを作り出すために開始タグに組み込まれている点に注意しなければならない。XML標準仕様との互換性は、場合によっては重要となることがある。好適な実施形態において、現在のタグ文字列の“/”文字は、開始タグと終了タグの名前がハッシュ関数から同じ数値が返されるために、次のタグの名前をハッシュにする前に、一般的には取り除かれる。XMLタグの例は、インターネット上の <http://www.w3.org/tr/2000/rec-XML-20001006.html> で入手できる『拡張可能なマークアップ言語 (XML) 1.0 (第2版) W3C勧告、2000年10月6日』の3.1節に見られるように、</Name Attribute>:によって示される。現在の記述では、“タグ”は、文脈によって、考慮している特定のタグの一部分又は全体のいずれかを参照する。一方、メモリ中では“/”文字 (終了タグを識別する) に相当する表現を保持することが必要となる場合もありうる。これは、以下のさまざまな方法で行われる。例えば、(i) 終了タグであることを示すために、終了タグのハッシュ数字の近傍で、“/”文字又はそれに相当する文字をメモリに「再挿入」すること、又は(ii) ハッシュタグの始まりか終わりの状態を示すブール値を用いること、又は、(iii) 完全にマッチングした場合、開始タグと終了タグの単純な和がゼロとなるように、終了タグのハッシュ値を負にすること、等が挙げられる。(iii)の場合、開始タグのハッシュは、本実施形態において、必要な終了タグのハッシュを作り出すために、単純な否定演算子によって変更される。(iii)は、符号ビットがハッシュアルゴリズム

```

133
133.326
133.371
133.-371
133.787
133.787.629      ->013307870629
133.787.-629     ->-013307870629
133.787
133.411
133.411.629
133.411.-629
133.-411
-133

```

【0079】 [5]において、ネストになったタグの構造は、[3]で示される形式から、一連の連結ハッシュタグに変換される。ここでは、各々の後続の低レベルの (すなわち、より深くネストされた) ハッシュタグの階層レベルは、前の上段の階層レベルに関連付けられる。これはハッシュで表されたDTDから、同様に解析された構造を用いて、簡単な数の比較を実行することができる。実際、[5]の各々の行は、[5]の行4及び行5

に影響を受けないという保証が必要である。事実上、

(iii)はブールフラグオプション(ii)とほとんど同じである。

【0075】さらに、構造化されたハッシュ番号が作られ、そこではネストになったタグ用のハッシュ番号は、第1のタグがネストになっているところではより高レベルのXMLタグであることを明示できる。従って、例えば、タグ123がタグ987の内部にネストになっているところでは、ネストになったタグ123のように示す代わりに、987.123のように示すことができる。この構造化されたハッシュ又は“拡張された”ハッシュにより、またがった構造の演算が減少する。すなわち、タグのペアの終了点を検索している間にメモリに保持しなければならないXML文書の量を減らすことによって解析の性能をさらに向上することができる。

【0076】拡張された表現は、ハッシュに基づく必要はなく、文字列や“数え上げ法”に基づくことができるという点にも注意しなければならない。数え上げ法は、タグの名前と数字の間に配置が定義され、目録表や索引が作られる処理である。数え上げ法の簡単な形式は、全てのタグの名前を単にリストにし、列記されたタグに番号を付けたものである。従って、例えば、“Shakespeare.banquo.quote”の形式の連結文字列は、3つの連結タグの文字型拡張表現を表す。

【0077】XML部[3]に対する構造化された同等のハッシュにされたマークアップの例は、負の終了タグのハッシュを用いた、以下の[5]で示される。

【0078】

[5]

で示されるように、出くわした一連のハッシュタグを連結して組み合わせさせた1つの数字で表される。この1つの数字は、元の入力タグの識別と関係を非常に圧縮された形で表し、それによって同様にハッシュで表されたDTDと非常に効率のよい比較ができる。数字のタグセットは、文書構造を表すために、非常に圧縮された形で用いることができることがわかる。妥当性チェックは、単にハッシュで表された開始タグのセットを用いるだけで

実行でき、各々のこのセットは、文書構造の各々の枝の最も深い、全体の、構造を表す点に注意が必要である。例えば、[5]の構造を最小限に表したものは、以下の

```
01330326
01330371
013307870629
013304110629
```

【0081】また、DTD又はXMLスキーマ構造も同じ方法によって表すことができる。

【0066】解析とハッシュが行われた入力文書からのタグセットと、解析とハッシュが行われたDTDの間の1つ又は複数の数値の比較は、XMLパーサの妥当性に通常必要とされる一連の文字列及び構造の比較を置き換える。DTD又はXMLスキーマによって定義されたどんな代替構造も、入力XML文書から生じたハッシュタグセット数字と後で比較するために用いる単一ハッシュタグセット数字にコード化できることがわかる。

【0082】図4は、例えば、DTD又はXMLスキーマのようなVRDに対して、マークアップ文書を検証するための処理600を示したものである。処理600は、検証されるマークアップ文書が開かれるステップ602から始まる。その後、ステップ604で、現在の拡張タグは、図5又は図6の各々のプロセッサ414又は505によってリセットされる。図4の説明において、“タグ”、“拡張タグ”、“テンポラリータグ”等は、各々のタグのハッシュ表現を指す。以下のステップ606では、テンポラリータグのルートは、各々のプロセッサ414又は505の1つによってリセットされ、その後、マークアップ文書中の次のタグが、ステップ608で識別される。その後、テストステップ610は、ステップ608で識別されたタグが開始タグであるかどうか判断する。テストステップ610のイベントでは、処理600は「YES」矢印に応じてステップ612へ向かう。ステップ612では、ステップ608で識別されたタグを、各々のプロセッサ414又は505のうち1つを用いた拡張タグに加える。次に、処理600はステップ612からステップ608へ戻る。

【0083】テストステップ610が、次のタグは開始タグではないと判断した場合、処理600は「NO」矢印に応じて、拡張タグが“0”かどうかを判断し文書のルートレベルを表すテストステップ614へ向かう。

“0”の値が検出された場合、処理600は「YES」矢印に応じて、文書の終わりに到達したかどうかを判断するテストステップ624へ向かう。もしそうでない場合には、処理600は「NO」矢印に応じてステップ606へ向かう。なお、ステップ614で“0”の値が検出されるのは、開始タグと終了タグの数が合わない構造等のように、整形性のない文書構造だからかもしれない。

【0084】テストステップ614が、拡張タグの値が

[6]のように示される。

【0080】

[6]

“0”でないことを判断した場合、処理は「NO」矢印に応じて、拡張タグがテンポラリータグのルート値と等しいかどうかを判断するテストステップ616へ向かう。そうでない場合には、処理600は「NO」矢印に応じて、各々のメモリ418及び506中に文書リスト中の拡張タグを格納するステップ618へ進む。一方、テストステップ616が、拡張タグがテンポラリータグのルートに等しいことを判断した場合、処理600は各々のプロセッサ414と505を用いて、「YES」矢印に応じて拡張タグから最も低い（すなわち最も深いネストである）タグを取り除くステップ620へ向かう。その後、処理600は、ステップ622で拡張タグをテンポラリータグのルートにコピーし、処理はステップ608へ戻る。

【0085】テストステップ624に戻る前に、これまで記述されていたように、処理600は正当性がチェックされるマークアップ文書へ向かう点に注意しなければならない。しかし、明確には記述されていないが、妥当性参照文書（VRD）に適用される同一の処理も存在し、この処理によって処理600により作り出される文書リストをテストできるVRDリストが作り出される。処理600及びVRDに向かうそれに相当する処理は、異なる時間に通常発生する。処理600は、検証される全ての文書に対して発生し、各々の特定の文書を検証するために拡張したハッシュ表現のリストを作り出す。VRDリストがステップ626の前に完成される場合、VRDリストは処理600と実質的には同時に作り出すことができる。一方、VRD処理はオフラインで実行することができ、結果のリストはステップ626の前に処理600に与えられる。

【0086】ステップ626に戻ると、前述のようにVRDリストが入手できるので、ステップ626は文書リスト中の全ての入力VRDリストに見つかるかどうかを判断する。もしそうである場合には、処理600は「YES」矢印に応じて、文書は妥当であると宣言するステップ628に向かう。一方、文書リストがVRDリストには見られない入力を持つ場合、処理600は「NO」矢印に応じて、文書が無効であると宣言するステップ630に向かう。

【0087】ステップ626でさらに詳しく述べられているように、前記記述では、文書リストの全ての入力とVRDリストの全ての入力を比較する。もう1つの処理は、ステップ616の後に、ステップ626と同様のス

テップでの完全なVRDリストに対して、各々の拡張タグをテストすることである。この処理のイベントでは、文書拡張タグがVRDリストに見つからなかった場合、これ以上不要なテストを行わずに、処理600がステップ630に直接進むことができる。しかし、拡張タグが見つかった場合、処理600はステップ620等に向かうことができる。この処理によって誤りを早く認識し、ただちに妥当性処理を終了することができる。VRDリ

01

02

03

-03

04

-04

-02

05

-05

-01

【0090】機能面から言えば、処理600はマークアップ文書の階層構造の最も深い枝の部分に降る。すなわち[7]の第1行にある“01”から[7]の第3行にある“03”へ向う。そして、その特定の枝の最も深い部分を表す拡張ハッシュ表現、すなわち“010203”を格納する。次に、この処理は、新たに探索すべき枝を示す別の開始タグ（この例の場合、[7]の5行目にある“04”）を見つけるまで、終了タグを捨て、枝を横切る。処理が新しい枝に降る場合、その処理は、階層のより高いレベルにある拡張ハッシュ表現を保存する。これらのハッシュ表現は、処理がこれらのレベルに戻るまで保存される。文書構造中の誤り（無効の文書、又は整形性されていない文書）は、これらのVRDにマッチングしない拡張ハッシュ表現を通常返す。ステップ620は、前の開始タグに対して検索された終了タグの整形性チェックを任意的に含んでもよく、その結果、文書に整形性があれば整形性マッチを提供する。前出の開始タグは、拡張ハッシュ表現の中で最低のタグである点に注意すべきである。例えば、DTD/XMLスキーマは、ステップ610で、ステップ620の拡張ハッシュ表現にある最低の開始タグにマッチングしない終了タグを返し、それによって整形性チェックにひっかかることになるかもしれない。

【0091】ステップ626でのテストは、一般的に、その文書のDTD用にリストになったハッシュ表現に対して、マークアップ文書構造の拡張ハッシュ表現をマッチングさせようとする。文書構造のハッシュ表現は、一般的にDTDリストにある最も深い構造表現のサブセットである。従って、妥当なXML文書は、対応するVRD又はDTD中に定義されている、あらゆる正当な構造的なネストのサブセットを含んでよい。従って、ステップ626における一般的なテストでは、DTDのより深い

ストが短ければさらに効率がアップする。完全な妥当性チェックが、前記方法で実行される場合、ステップ628は考慮している文書が妥当であることを示す。

【0088】前述の妥当性の方法をさらに説明するために、以下の構造部（ここでは開始タグが“01”から“05”であり、対応する終了タグはそれぞれ“-01”から“-05”である）を考える。

【0089】

[7]

ハッシュ表現と、文書のより浅いハッシュ構造表現との比較を含む。従って、例えば、XML文書からの拡張ハッシュ表現“0123”は、対応するDTDからのハッシュ表現“01230456”と比較した場合、「妥当性がある」と評価される。

【0092】図4に示される妥当性処理600は、高速だが不完全な妥当性チェックを用いて、最も深くネストになった部分等の、文書構造中のより複雑な部分をチェックするように、最適化されうる。従って、任意の性能特性を有する特定の妥当なパーサを実行するために、速度と妥当性の感度の最適な組み合わせを選択できる。

【0093】また、妥当性方法600を変更して「標準的な」整形性チェックの少なくとも一部分を実行することもできる。従って、例えばステップ620において、拡張タグ表現の中で、最も低い階層にあるタグ表現のハッシュに対して、終了タグのハッシュ表現をチェックすることができる。前記表現が、同じオリジナルのタグであると適切に識別されなかった場合、その文書は整形性がなく、回復や誤り処理が実行される。

【0094】前記方法は、構造チェックとは別にまたは一緒に、構造内で定義された属性のハッシュ表現を含むように拡張できる。

【0095】この妥当性及び整形性チェックの方法を、別々の処理における入力文書や、文書構造及び内容を解析する処理に対して、適用できることは明らかである。従って、例えば、方法600は、中央演算処理装置(CPU)サイクル及びメモリサイズが特に大きな制約を受けていない環境においても、妥当性及び整形性チェックを、効率良く、高速に実行できるようにするために、最適化することができる。そのようなシステムで別々のチェックを実行する利点として、高度に最適化されたチェックを用いて、「妥当性のない」文書を直ちに廃棄する

ことができるという事実が挙げられる。これは、少なくとも妥当でない文書の一部分にかかる時間と処理を大幅に節約することができる。それによって、例えば、(i) 文章が妥当でないことを見つけるためだけに、文章を解析して完全なDOMツリー表現にし、妥当性チェックを実行すること、又は(ii)妥当でない文書の第2の部分を検出する前に、文書の第1の(妥当な)部分の処理を更に開始すること(文書の第1部分の処理がそれによって無駄になっている)を防ぐことができる。もう1つの利点は、高速な妥当性チェックを用いて文書に妥当性がないとわかった後、次のジョブの処理を直ちに始められる点である。

【0096】「不完全な」ハッシュ処理、すなわち、各々の英数字の入力文字列に対して単一の数字を作り出す保証がないハッシュ処理にも、適した場合があり、特に、XMLタグ文字列の最大長が制限されている、又はある確率で少なくとも制限されている場合に適している。さらに、XMLタグ文字列セットが限られた数の文字順列に制限されるか、又は限られた数の文字順列に制限される可能性がある場合、適切に動作するために不完全なハッシュ処理が設計され、選択される。

【0097】文書構造の数的表現のための通信規格、又は、代替の公用及び私的フォーマットは、ハッシュアルゴリズムの使用に基づいて、定義され、記述される。この技術によって、冗長性のために大量のデータ伝送を通常伴う、XMLデータの伝送にメリットとなる圧縮形式及び人間が読み易いASCII形式が可能となる。人間の読み易さを保持するか、しないかに対して、様々な選択肢が存在する。例えば、(完全な)ハッシュを、他の圧縮形式と組み合わせることが挙げられる。これは、XMLファイル中の様々な要素タイプにそれぞれ適用される例えば、XML文字列タグを、完全なハッシュアルゴリズムから得られる、ユニークで、人間が読みやすい数字に、置き換えることができる。また、ハッシュではない構文要素及び他の要素は、処理又は装置間で、伝送用の非損失圧縮技術によって圧縮することができ、それによって、伝送データ量が減少する。

【0098】前段落で述べたように、必要なところに、逆ハッシュアルゴリズム又は可逆ハッシュアルゴリズムを参照したり、含んだりすることができる。これは、例えば、ハッシュされる前の、伝送された、マークアップ文書から、表示したり、ラベリングしたりする目的で、1つ以上のマークアップタグを人間が読み易い文字列にデコード、又は解読するために、そのようなアルゴリズムが必要とされる場合に用いられる。ここでは、別に、解析や誤りチェックのためにそのようなことを行う必要はない。可逆ハッシュアルゴリズム、又は逆ハッシュアルゴリズムのもう1つ利用法は、マークアップタグ又は他のデータを解読することによって、伝送されたマークアップ文書について制限された機能や特徴が使用できる

ようになることである。可逆アルゴリズム、又は逆アルゴリズムは、マークアップ文書の送信部と受信部のマッチングに使用することができる。可逆ハッシュアルゴリズム、又は逆ハッシュアルゴリズムは、受信部に既に含まれているので伝送はされないが、マークアップ文書で参照される場合もある。可逆ハッシュアルゴリズム、又は可逆ハッシュアルゴリズムには、(i)完全非損失アルゴリズム、及び(ii)ハフマン符号化、が例として挙げられる。

【0099】上記機構は、以下の条件の1つ以上が当てはまる場合には、あらゆるマークアップ言語に対して、非常に有利な立場で適用することができる。その条件とは、すなわち、(i)マークアップ言語はタグ名の定義(例えば、XML、DTD、CSS、XSL等)を許す、(ii)タグ名は大きな文字コードテーブル(例えば、UTF-16)を使用し、及び/又は、タグ名の長さは一般的にそのハッシュ表現よりも短くない、(iii)マークアップ文書を用いる又は受け取る使用アプリケーションは、マークアップ文書、XMLスキーマ又はDTDの中に最低でも1つのネストの階層レベルを持った、複雑な構造表現を通常必要とする、(iv)入力マークアップ文書は、何らかのチェック形式、通常は整形性又は妥当性チェックが必要である、(v)マークアップパーサ、及び/又は、アプリケーションは、メモリ容量(例えば、組み込み式又は低価格CPUシステム)又はメモリ管理(例えば、仮想メモリがないシステム、又はダイナミックメモリが割り当てられていないシステム)に厳しい制限がある、(vi)複雑で、高度にネストなる可能性があるマークアップ文書上で素早く動作する必要があるマークアップパーサ及び/又は、アプリケーションである。

【0100】本明細書で示されたマークアップ言語文書の解析方法は、マークアップ言語文書の解析用関数、又はサブ関数を実行する、1つ以上の集積回路等の専用ハードウェアで実行できる。そのような専用ハードウェアは、画像プロセッサ、デジタル信号プロセッサ、又は1つ以上のマイクロプロセッサ及び関連メモリを有していてもよい。マークアップ言語文書の解析方法は、一方、専用組み込みコンピュータシステム400を用いて、実行することができる。そのようなシステムは図5で示される。このシステムにおいて、図3A、図3B、図3C、及び図4の処理は、組み込みコンピュータシステム400中で実行されるアプリケーションプログラム等のソフトウェアとして実行されてもよい。コンピュータシステム400は、典型的には、プリンタ(図には示されていない)等のエンドシステムに組み込まれ、プリンタ中のプリンタエンジン402を動作させる。特に、組み込みコンピュータによって実行されるソフトウェアの命令によって、マークアップ言語の解析方法のステップが行われる。そのソフトウェアは、読み出し専用メモリ(ROM)418、ランダムアクセスメモリ(RAM)41

8、又は他のタイプのメモリ（示されていない）を含んだコンピュータ可読媒体に収納されてもよい。そのソフトウェアは、製造時に組込みコンピュータにロードされるか、或は、その場でソフトウェアのアップグレードが行われる。

【0101】組込みコンピュータシステム400は、コンピュータモジュール410、パラメタ設定用のスイッチモジュール422等の入力装置、ジョブ状態を示す液晶表示器（LCD）等の出力装置、プリンタエンジン402で構成される。組込みコンピュータ400は、通常、物理的にプリンタ（示されていない）に組み込まれる。コンピュータネットワーク404に接続された他のコンピュータ（示されていない）で起こった印刷ジョブは、入出力（I/O）インターフェース408への接続404によって、組み込みコンピュータ400に送られる。

【0102】組込みコンピュータモジュール410は、プロセッサユニット414及びメモリユニット418、更に、例えば、半導体ランダムアクセスメモリ（RAM）、読み込み専用メモリ（ROM）、スイッチモジュールとLCDインターフェース416を含む入出力（I/O）インターフェース、プリンタエンジン402とネットワーク406用の入出力インターフェース408で構成される。組込みコンピュータ410の構成要素408及び構成要素414～418は、通常相互接続されたバス412を介して交信しており、これは関連技術で知られる組込みコンピュータシステム410の従来動作モードのようになる。一般的に、この機構に関するプログラムは、メモリ418に格納され、プログラム動作時にプロセッサ414がプログラムを読み込んで制御する。

【0103】また、マークアップ言語文書の解析方法は、図6で示されるような、従来型汎用コンピュータシステム500を用いて実行することができる。このシステムにおいて、図3A、図3B、図3C、及び図4の処理は、コンピュータシステム500中で動作するアプリケーションプログラム等のソフトウェアとして実行されてもよい。このアプリケーションは、例えば、ハッシュが、ネットワークを介したコンピュータ間の通信規格として用いられる場合に役立つ。図6は、現在考慮している通信コンピュータの一例を示したものである。

【0104】特に、マークアップ言語文書の解析方法のステップは、コンピュータによって実行されるソフトウェアの命令によって行われる。そのソフトウェアは2つの分離部品に分けられてもよい。すなわち、1つは解析方法を実行する部分で、もう1つは後者とユーザの間のユーザインターフェースを管理する部分である。例えば、そのソフトウェアは、以下に示す記憶装置を有するコンピュータ可読媒体に格納されてもよい。そのソフトウェアはコンピュータ可読媒体からコンピュータにロードされ、次にコンピュータにより実行される。そのよう

なソフトウェアを有するコンピュータ可読媒体、又は前記媒体に記録されたコンピュータプログラムは、コンピュータプログラム製品である。このコンピュータプログラム製品のコンピュータでの使用は、本発明の実施形態に応じて、マークアップ言語文書を解析に有利な装置で実行されるのが望ましい。

【0105】コンピュータシステム500は、コンピュータモジュール501、キーボード502及びマウス503等の入力装置、プリンタ515及びディスプレイ装置514を含む出力装置で構成される。変調・復調（モデム）送受信装置516は、例えば、電話線521又は他の機能媒体を介して接続可能な通信ネットワーク520へ双方向で通信するために、コンピュータモジュール501で用いられる。モデム516は、インターネットや他のネットワークシステム、例えば、ローカルエリアネットワーク（LAN）又はワイドエリアネットワーク（WAN）、コンピュータ500と通信している他のパーソナルコンピュータ（PC）522等へアクセスできるようにするために用いられる。

【0106】コンピュータモジュール501は、典型的には、少なくとも1つのプロセッサユニット505、メモリユニット506を含み、例えば、半導体ランダムアクセスメモリ（RAM）、読み込み専用メモリ（ROM）、ビデオインターフェース507を含んだ入力・出力（I/O）インターフェース、キーボード502及びマウス503用の入力・出力インターフェース513、及びオプションとしてのジョイスティック（図で示されていない）、モデム516のためのインターフェース508で構成される。

【0107】記憶装置509が設けられており、通常、ハードディスクドライブ510及びフロッピー（登録商標）ディスクドライブ511を含む。また、磁気テープドライブ（不図示）を用いてもよい。CD-ROMドライブ512は、非揮発性のデータ源として通常備えられる。コンピュータモジュール501の構成要素505～513は、相互接続されたバス504を介して通常交信し、これは関連技術で知られるコンピュータシステム500の従来動作モードのようになる。本実施形態において実行できるコンピュータには、IBM PC、IBM PC互換機、及びSun Sparcstations又はそこから発展した同様のシステムが例に挙げられる。

【0108】典型的には、本実施形態におけるアプリケーションプログラムは、ハードディスクドライブ510上に格納され、プログラム動作時にプロセッサ505がプログラムを読み込んで制御する。ネットワーク520から取り出されたプログラム及びあらゆるデータの間記憶装置は、場合によってはハードディスクドライブ510と連動して、半導体メモリ506を用いて実現してもよい。場合によっては、アプリケーションプログラムは、CD-ROM又はフロッピーディスクにエンコード

してユーザに供給されてもよく、対応するドライブ 512 又は 511 を介して読み込まれるか、代わりに、モデム装置 516 を介してネットワーク 520 上で PC 522 からユーザによって読まれてもよい。

【0109】さらに、磁気テープ、ROM、集積回路、磁気光ディスク、ラジオ又はコンピュータモジュール 501 及び別の装置間の赤外線トランスミッションチャンネル、PCMCIA カード等のコンピュータ可読カード、電子メールの送信及びウェブサイトやそれと同様のものを有するインターネット及びイントラネットを含む他のコンピュータ可読媒体から、コンピュータシステム 500 にソフトウェアをロードすることもできる。上記は、コンピュータ可読媒体に関連した例に過ぎない。発明の範疇や技術的思想から外れない限り、他のコンピュータ可読媒体は、実行してもよい。

【0110】（産業上の利用分野）以上から、本発明の実施形態は、コンピュータ及びデータ処理産業に適用できることは明らかである。上記は、本発明の実施形態のいくつかを示したに過ぎず、発明の範囲や意図から外れない限り、修正及び/又は変更を行うことができる。上記実施形態は、例に過ぎず、限定的なものではない。

【図面の簡単な説明】

【図 1 A】本発明の好適な実施形態に係る XML パーサシステムのブロック図である。

【図 1 B】本発明の好適な実施形態に係る XML パーサシステムのブロック図である。

【図 2 A】選択的な整形性チェック及び/又は妥当性チェックを含む、従来技術における SAX パーサの方法ステップを示すフローチャートである。

【図 2 B】選択的な整形性チェック及び/又は妥当性チェックを含む、従来技術における SAX パーサの方法ステップを示すフローチャートである。

【図 3 A】図 2 A 及び図 2 B の SAX パーサを改良した機構を示す図である。

【図 3 B】図 2 A 及び図 2 B の SAX パーサを改良した機構を示す図である。

【図 3 C】図 2 A 及び図 2 B の SAX パーサを改良した機構を示す図である。

【図 4】DTD 又は XML スキーマ等の参照文書に対する、文書の妥当性の処理について示す図である。

【図 5】改良された SAX パーサの機構を実行できる専用組込みコンピュータのブロック図である。

【図 6】改良された SAX パーサの機構を実行できる汎用コンピュータを示す図である。

【図 1 A】

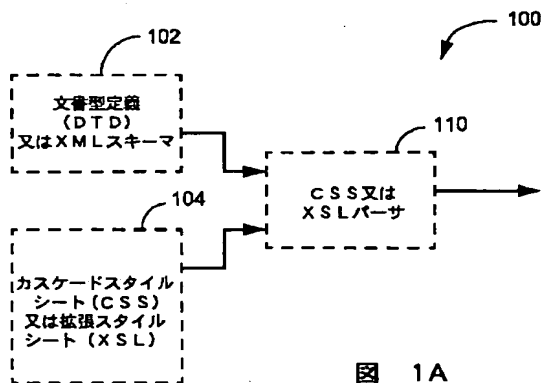


図 1A

【図 1 B】

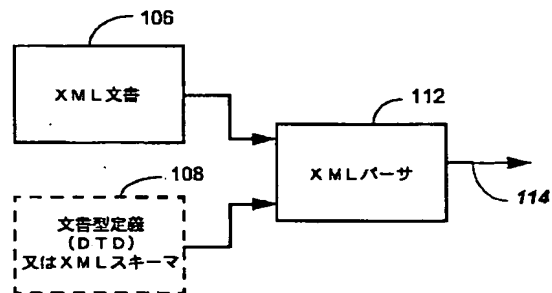
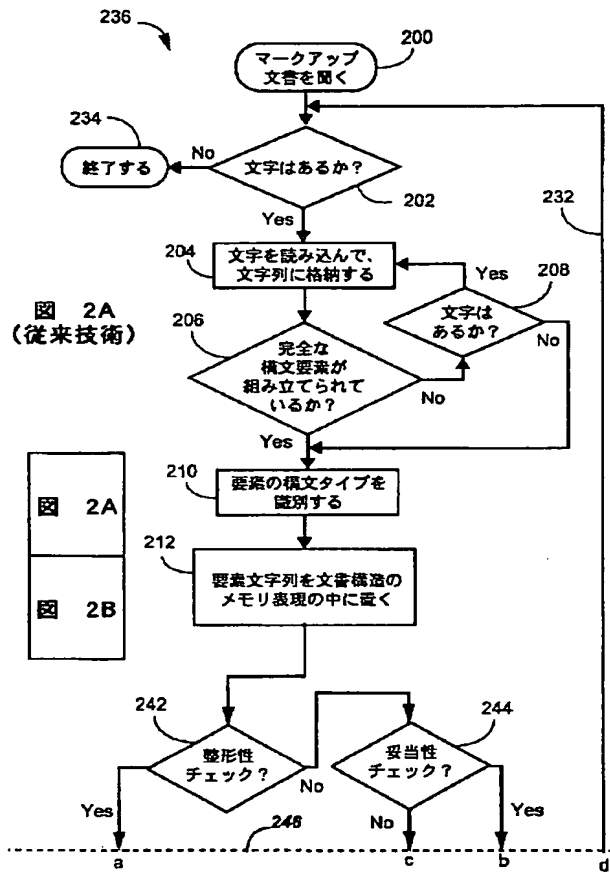
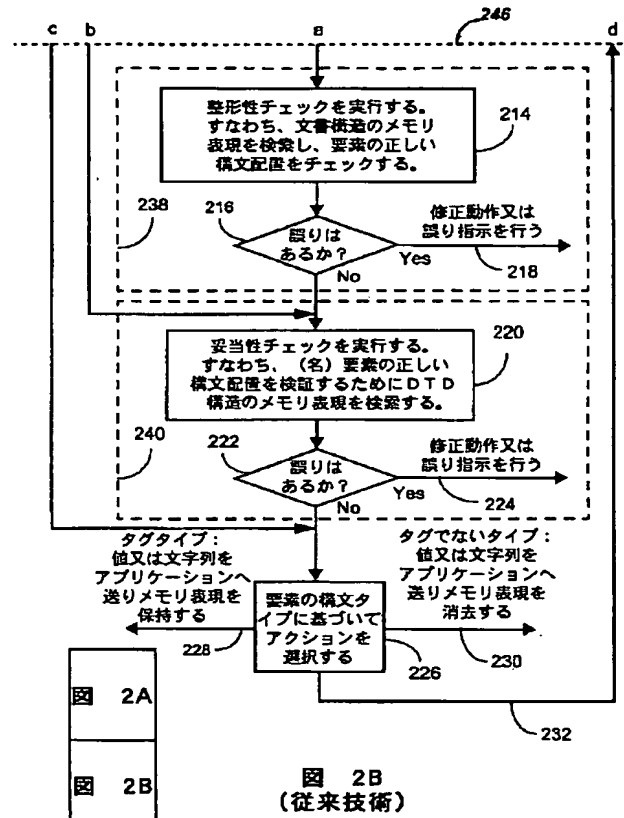


図 1B

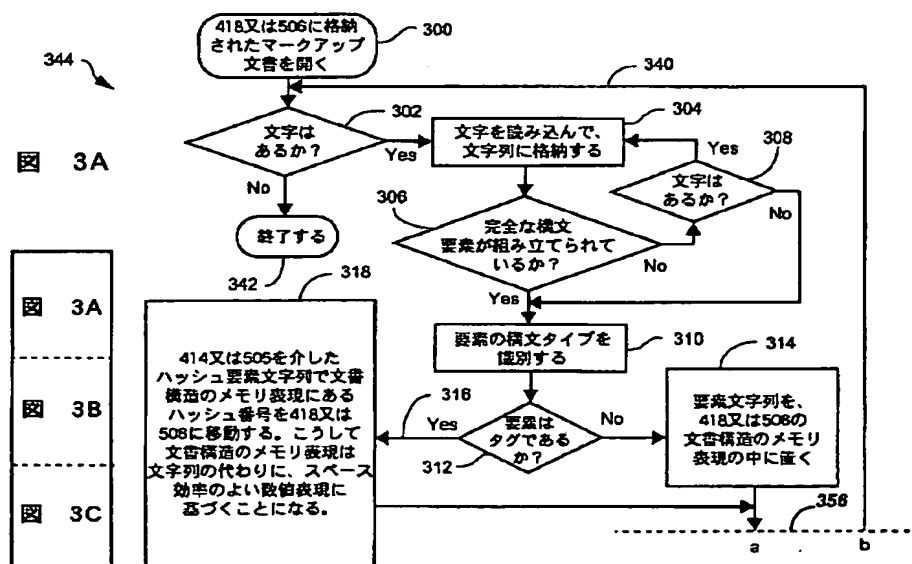
【図 2 A】



【図 2 B】

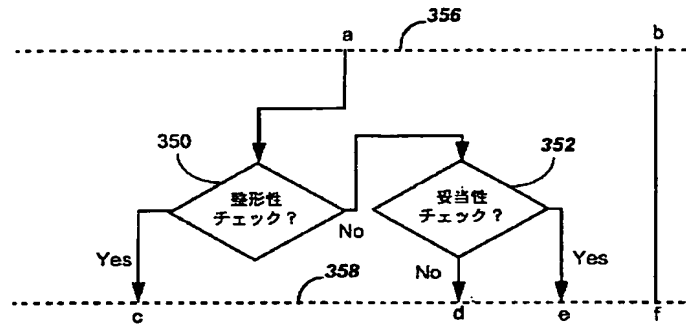
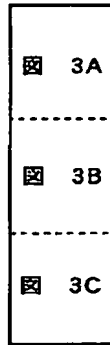


【図 3 A】

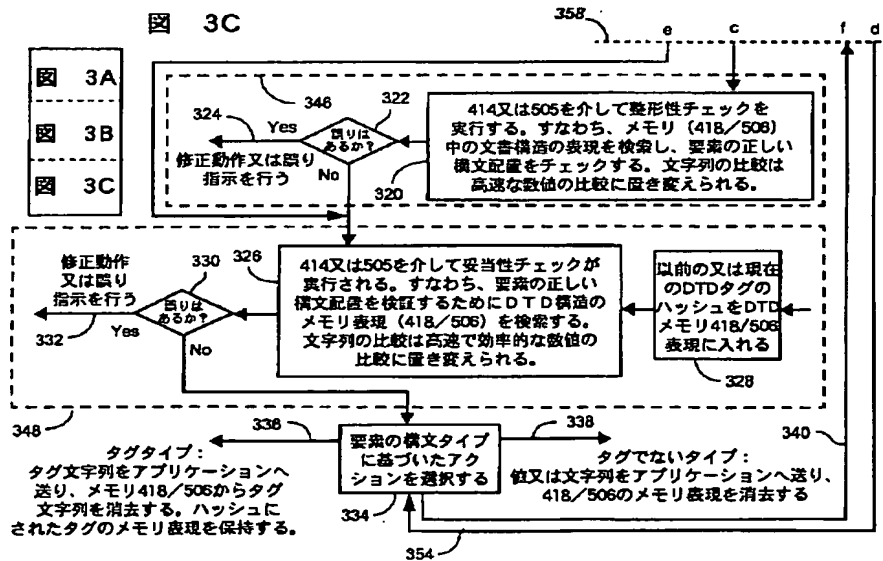


【図 3 B】

図 3B



【図 3 C】



【図4】

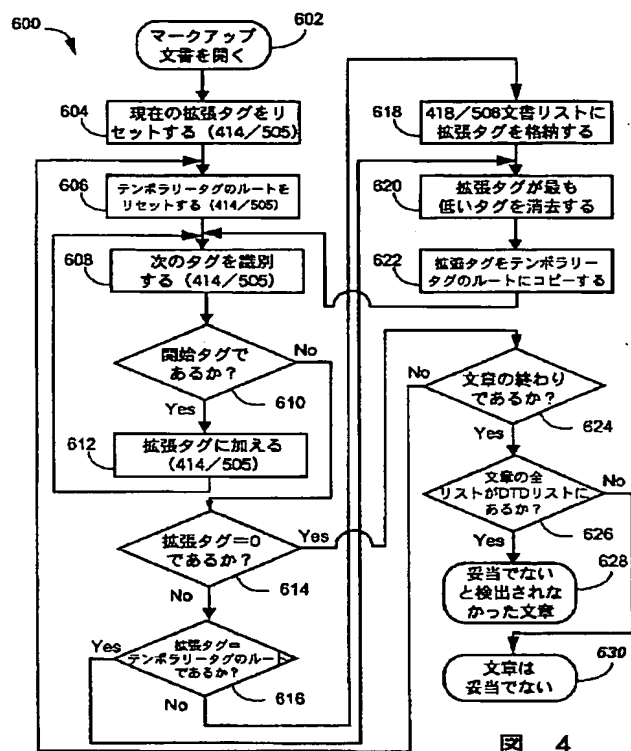


図 4

【図5】

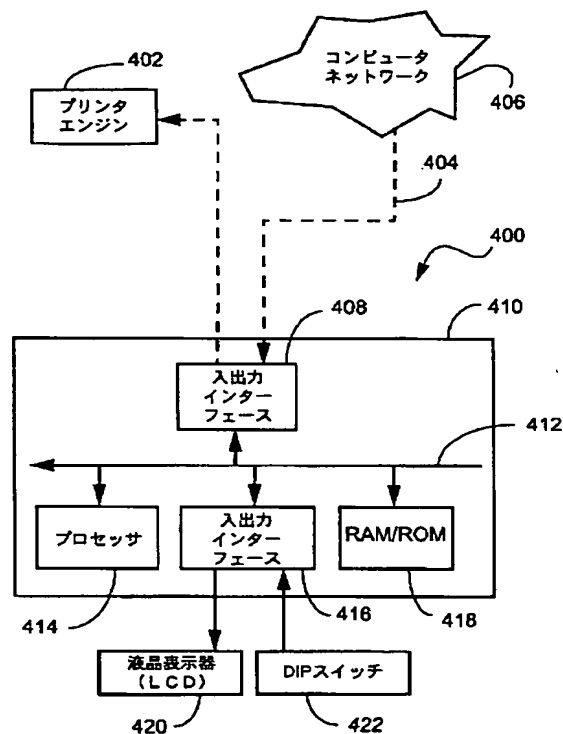


図 5

【図6】

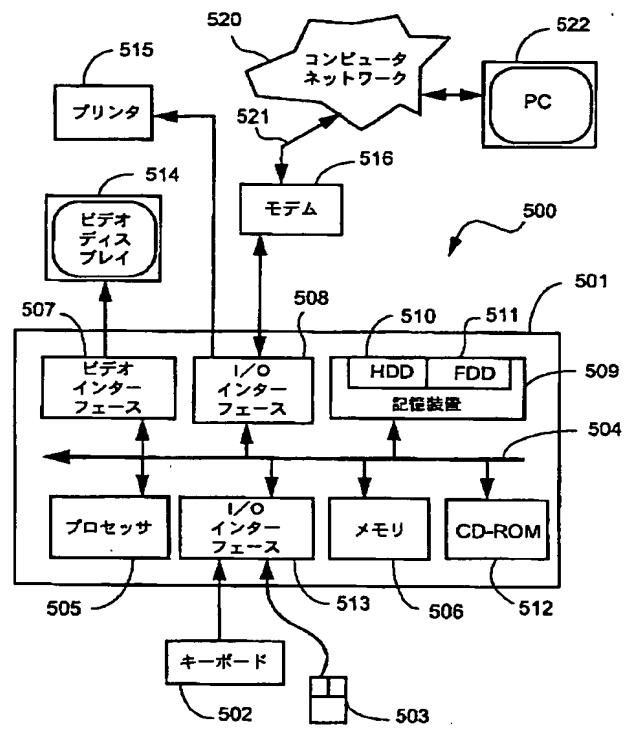


図 6

【外国語明細書】**1. Title of the Invention****HASH COMPACT XML PARSER****2. Claims:**

1. A method of parsing a markup language document comprising syntactic elements, said method comprising, for one of said syntactic elements, the steps of:
identifying a type of the element;
processing the element by determining a hash representation thereof if said type is a first type; and
augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.
2. A method according to claim 1, wherein said parsing is event-based parsing.
3. A method according to claim 1, wherein said hash representation is determined using one of:
a hash algorithm;
a first reference to said hash algorithm dependent upon an associated Universal Reference Indicator;
a second reference to said hash algorithm dependent upon an associated namespace; and
a third reference to said hash algorithm dependent upon an associated Extended Markup Language declaration;
4. A method according to claim 2, wherein said first type is one of:
one of a structural element and a part thereof;
a definition of said structural element;

a declaration of said structural element; and
a match for said structural element.

5. A method according to claim 4, wherein said structural element is a tag.

6. A method according to claim 2, wherein the hash representation is a unique code for said one syntactic element, said element having less than a first number of characters.

7. A method according to claim 2, wherein the hash representation is not a unique code for said one syntactic element, said element being constrained, to a probability level, in terms of at least one of (i) a number of characters in the element and (ii) a permissible number of permutations of characters in the element.

8. A method according to claim 6, wherein said code comprises numeric characters.

9. A method according to claim 2, wherein said processing step comprises a sub-step of:

determining an extended hash representation of both (i) said one syntactic element being a first instance of said first type, and (ii) another syntactic element being a second instance of said first type, within which said first instance, said second instance is nested.

10. A method according to claim 1 comprising, for another one of said syntactic elements, further steps of:

identifying a type of the other element; and if the type of the other element is equivalent to said first type:

(i) processing the other element to thereby determine a second hash representation thereof; and

(ii) augmenting said at least partial structural representation of the document using the second hash representation, wherein:

said processing and said second processing ensure that if a first relationship exists between the one element and the other element, then a second relationship which is representative of the first relationship, exists between the hash representation of the one element and the hash representation of the other element.

11. A method according to claim 10, wherein:

the one element is a start tag;

the other element is an end tag;

the hash representation of the one element is a corresponding hashed start tag,

and;

the second hash representation of the other element is a corresponding hashed end tag.

12. A method according to claim 11, wherein:

the end tag is a first modification of the start tag; and

the hashed end tag is a second modification of the hashed start tag, said second modification being representative of the first modification.

13. A method according to claim 12, wherein:

the end tag is the same as the start tag apart from having a distinguishing character incorporated therein; and

the hashed end tag is at least one of:

the same as the hashed start tag;

the same as the hashed start tag apart from having a distinguishing character incorporated therein; and

the hashed start tag having been processed by an operator.

14. A method according to claim 12, wherein:

the one and the other element comprise respectively a start tag and an end tag, being a first pair of tags;

corresponding hashed start and end tags for said first pair of tags are incorporated into the partial structural representation of said document;

the document further includes a second pair of tags comprising a respective start tag and end tag, said second pair of tags being nested within said first pair of tags in said document, said method comprising further steps of:

processing said second pair of tags to form corresponding second hashed start and end tags;

augmenting said at least partial structural representation of the document using said corresponding second hashed start and end tags so that said second hashed start and end tags indicate a nesting in relation to said hashed start and end tags for the first pair of tags which is equivalent to the nesting of said second pair of tags within said first pair of tags.

15. A method according to claim 14 comprising, prior to said augmenting step, a further step of:

concatenating the first hashed start tag with the second hashed start tag, and concatenating the first hashed end tag with the second hashed end tag, to thereby form respective extended hashed start and end tags for said second pair, wherein:

said augmenting step is performed using said respective extended hashed start and end tags for said second pair, and:

said extended hashed start and end tags indicate a nesting in relation to said hashed start and end tags for the first pair of tags which is equivalent to the nesting of said second pair of tags within said first pair of tags.

16. A method according to claim 1, wherein the augmenting step[†] is succeeded by a well-formedness checking step against a syntactic rule, said well-formedness checking step comprising checking said at least partial structural representation of the markup language document against the syntactic rule by numerically

comparing corresponding hashed representations of elements in said at least partial structural representation of the markup language document

17. A method according to claim 16, wherein said numerically comparing step is succeeded by a further step of:

string-comparing, in accordance with said syntactic rule, corresponding non-hashed representations of elements not of said first type.

18. A method according to claim 1, wherein said first type is one of:
one of a structural element and a part thereof;
a definition of said structural element;
a declaration of said structural element; and
a match of said structural element.

19. A method according to claim 18, wherein said structural element is a tag.

20. A method according to claim 14, comprising a further step of:
checking the well-formedness of said at least partial structural representation of the document against a syntactic rule.

21. A method according to claim 20, wherein the syntactic rule relates to proper nesting of tags and said checking step comprises sub-steps of:

performing a numerical comparison across hashed tags in said at least partial structural representation of the document to thereby identify said first hashed start and end tags and said second hashed start and end tags; and

verifying that the second hashed start and end tags indicate a proper nesting in relation to said first hashed start and end tags.

22. A method according to claim 21, wherein the numerical comparison is followed by a further step of:

performing a string comparison, in accordance with said syntactic rule, across non-hashed parts of respective tags in said at least partial structural representation of the document.

23. A method according to claim 15, comprising a further step of:

checking the well-formedness of said at least partial structural representation of the document against a syntactic rule.

24. A method according to claim 23, wherein the syntactic rule relates to proper nesting of tags and said checking step comprises sub-steps of:

performing a numerical comparison across hashed tags in said at least partial structural representation of the document to thereby identify said first hashed start and end tags and said extended hashed start and end tags; and

verifying that the extended hashed start and end tags indicate a proper nesting in relation to said first hashed start and end tags.

25. A method according to claim 24, wherein the numerical comparison is followed by a further step of:

performing a string comparison across non-hashed parts of respective tags in said at least partial structural representation of the document.

26. A method according to claim 16, wherein the well-formedness checking step is one of (a) succeeded by, (b) included in, and (c) replaced by a validation step against a validation reference document VRD, said validation step comprising sub-steps of:

(a) processing the VRD, said processing comprising, for a syntactic element in the VRD, sub-sub-steps of:

(i) identifying a type of said syntactic element of the VRD; and

(ii) processing the syntactic element by determining a hash representation thereof if said type is said first type; and

(b) checking said at least partial structural representation of the markup language document against the processed VRD, said checking comprising a sub-sub-step of numerically comparing corresponding hashed representations of elements.

27. A method of validating a markup language document against a VRD, said method comprising steps of:

(a) processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said processing comprising steps of:

(i) determining a hierarchy position of said document tag;

(ii) determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(b) processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said processing comprising steps of:

(i) determining a hierarchy position of said tag;

(ii) determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) storing said extended hashed representation of said tag in a list; and

(c) validating said markup language document if said extended hashed representation of said document tag is one of found in said list and is a valid subset of a member of said list.

28. A method of validating a markup language document against a VRD, said method comprising steps of:

(a) processing said VRD, for each structural element identified therein, said processing comprising steps of:

- (i) determining syntactic attributes of said structural element;
- (ii) determining a hashed representation of said structural element; and
- (iii) storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) processing the markup language document, for each document structural element identified therein, said processing comprising steps of:

- (i) determining syntactic attributes of said document structural element;
- (ii) determining a hashed representation of said document structural element; and
- (iii) storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

29. A method according to claim 26, wherein said numerically comparing step is succeeded by a further step of string-comparing corresponding non-hashed representations of elements not of said first type.

30. A method according to claim 26, wherein said first type is one of:
one of a structural element and a part thereof;
a definition of said structural element;
a declaration of said structural element; and
a match of said structural element.

31. A method according to claim 30, wherein said structural element is a tag.

32. A method of encoding a markup language document comprising syntactic elements, said method comprising, for one of said syntactic elements, steps of:

identifying a type of the syntactic element; and

processing the syntactic element by one of:

- (i) determining a hash representation thereof if said type is a first type;
- (ii) determining a compressed representation thereof if said type is not a first type; and
- (iii) retaining the syntactic element.

33. A method of decoding a markup language document comprising encoded syntactic elements, said method comprising, for one of said encoded syntactic elements, steps of:

identifying a type of the encoded syntactic element;

processing the encoded syntactic element by at least one of:

- (i) determining an inverse hash representation thereof if said type is a first type; and
- (ii) determining a decompressed representation thereof if said type is not a first type; and
- (iii) retaining the encoded syntactic element.

34. An apparatus for parsing a markup language document comprising syntactic elements, said apparatus comprising:

identifying means for identifying a type of the element;

processing means for processing the element by determining a hash representation thereof if said type is a first type; and

augmenting means for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

35. An apparatus for validating a markup language document against a VRD, said apparatus comprising:

(a) means for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said means comprising:

(i) means for determining a hierarchy position of said document tag;

(ii) means for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) means for storing said extended hashed representation of said document tag if said document tag is more deeply nested than an extended hashed representation of a previous document tag;

(b) means for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said means comprising:

(i) means for determining a hierarchy position of said tag;

(ii) means for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) means for storing said extended hashed representation of said tag in a list; and

(c) means for establishing whether said extended hashed representation of said document tag is one of to be found in said list, and is a valid subset of a member of said list, thereby validating said markup language document.

36. An apparatus for validating a markup language document against a VRD, said apparatus comprising:

(a) means for processing said VRD, for each structural element identified therein, said means comprising:

(i) means for determining syntactic attributes of said structural element;

(ii) means for determining a hashed representation of said structural element; and

(iii) means for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) means for processing the markup language document, for each document structural element identified therein, said means comprising:

(i) means for determining syntactic attributes of said document structural element;

(ii) means for determining a hashed representation of said document structural element; and

(iii) means for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) means for comparing syntactic attributes and hashed representations of said each document structural element in the structural representation of the document to corresponding syntactic attributes and hashed representations in said structural representation of said VRD to thereby establish validity of the markup language document.

37. An apparatus for encoding a markup language document comprising syntactic elements, to form an at least partial structural representation of the document, said apparatus comprising:

means for identifying a type of the syntactic element; and

means for processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

38. An apparatus for decoding a markup language document comprising encoded syntactic elements, said apparatus comprising:

means for identifying a type of the encoded syntactic element;

means for processing the encoded syntactic element by at least one of:

- (i) determining an inverse hash representation thereof if said type is a first type;
- (ii) determining a decompressed representation thereof if said type is not a first type; and
- (iii) retaining the encoded syntactic element.

39. A computer program which is configured to make a computer execute a procedure to parse a markup language document comprising syntactic elements, said program comprising:

- code for identifying a type of an element;
- code for processing the element by determining a hash representation thereof if said type is a first type; and
- code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

40. A computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

- (a) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:
 - (i) code for determining a hierarchy position of said document tag;
 - (ii) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and
 - (iii) code for storing said extended hashed representation of said document tag if said tag is more deeply nested than a previous document tag;

(b) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said code comprising:

- (i) code for determining a hierarchy position of said tag;
- (ii) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and
- (iii) code for storing said extended hashed representation of said tag in a list; and

(c) code for validating said markup language document if said extended hashed representation of said document tag is one of found in said list, and is a valid subset of a member of said list.

41. A computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing said VRD, for each structural element identified therein, said code comprising:

- (i) code for determining syntactic attributes of said structural element;
- (ii) code for determining a hashed representation of said structural element; and
- (iii) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) code for processing the markup language document, for each document structural element identified therein, said code comprising:

- (i) code for determining syntactic attributes of said document structural element;
- (ii) code for determining a hashed representation of said document structural element; and
- (iii) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) code for validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

42. A computer program which is configured to make a computer execute a procedure to encode a markup language document comprising syntactic elements, said program comprising:

code for identifying a type of the syntactic element; and

code for processing the syntactic element by one of:

- (i) determining a hash representation thereof if said type is a first type;
- (ii) determining a compressed representation thereof if said type is not a first type; and
- (iii) retaining the syntactic element.

43. A computer program which is configured to make a computer execute a procedure to decode a markup language document comprising encoded syntactic elements, said program comprising:

code for identifying a type of the encoded syntactic element;

code for processing the encoded syntactic element by at least one of:

- (i) determining an inverse hash representation thereof if said type is a first type; and
- (ii) determining a decompressed representation thereof if said type is not a first type; and
- (iii) retaining the encoded syntactic element.

44. A computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to parse a markup language document, said program comprising:

code for identifying a type of the element;

code for processing the element by determining a hash representation thereof if said type is a first type; and

code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

45. A computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said document tag;

(ii) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) code for storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(b) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said tag;

(ii) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) code for storing said extended hashed representation of said tag in a list; and

(c) code for validating said markup language document if said extended hashed representation of said document tag is one of found in said list and is a valid subset of a member of said list.

46. A computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing said VRD, for each structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said structural element;

(ii) code for determining a hashed representation of said structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said document structural element;

(ii) code for determining a hashed representation of said document structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) code for validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

47. An at least partial structural representation a markup language document comprising syntactic elements, said at least partial representation having been produced by a method comprising, for one of said syntactic elements, the steps of:

identifying a type of the element;

processing the element by determining a hash representation thereof if said type is a first type; and

augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

48. An apparatus for parsing a markup language document comprising syntactic elements, said apparatus comprising:

a processor;

a memory for storing (i) the document, and (ii) a program which is configured to make the processor execute a procedure to parse the document;

said program comprising:

(i) code for identifying a type of an element;

(ii) code for processing the element by determining a hash representation thereof if said type is a first type; and

(iii) code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

49. An apparatus for validating a markup language document comprising syntactic elements against a VRD comprising syntactic elements, said apparatus comprising:

(a) a processor;

(b) a memory for storing (i) the document, (ii) said VRD, and (iii) a program which is configured to make the processor execute a procedure to validate the document;

(c) said program comprising:

(ca) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(caa) code for determining a hierarchy position of said document tag;

(cab) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(cac) code for storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(cb) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said means comprising:

(cba) code for determining a hierarchy position of said tag;

(cbb) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(cbc) code for storing said extended hashed representation of said tag in a list; and

(cc) code for establishing whether said extended hashed representation of said document tag is one of to be found in said list, and is a valid subset of a member of said list, thereby validating said markup language document.

50. An apparatus for validating a markup language document containing syntactic elements against a VRD containing syntactic elements, said apparatus comprising:

(a) a processor;

(b) a memory for storing (i) the document, (ii) said VRD, and (iii) a program which is configured to make the processor execute a procedure to validate the document;

(c) said program comprising:

(ca) code for processing said VRD, for each structural element identified therein, said code comprising:

(caa) code for determining syntactic attributes of said structural element;

(cab) code for determining a hashed representation of said structural element; and

(cac) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(cb) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(caa) code for determining syntactic attributes of said document structural element;

(cab) code for determining a hashed representation of said document structural element; and

(cac) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(cc) code for comparing syntactic attributes and hashed representations of said each document structural element in the structural representation of the document to corresponding syntactic attributes and hashed representations in said structural representation of said VRD to thereby establish validity of the markup language document.

51. A method of validating a markup language document against a VRD, said method comprising steps of:

determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

storing said first extended hashed representation(s) in a VRD list;

determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

52. A method according to claim 51, wherein said syntactic element of said first type is one of:

one of a structural element and a part thereof;

a definition of said structural element;

a declaration of said structural element; and

a match for said structural element.

53. A method according to claim 52, wherein said structural element is a tag.

54. A method according to claim 51, wherein:

said most deeply nested syntactic elements(s) in the VRD are syntactic elements(s) which are most deeply nested within one of the global structure of the VRD and a local sub-structure of the VRD; and

said most deeply nested syntactic element in the markup language document is a syntactic element which is most deeply nested within one of the global structure of the markup language document and a local sub-structure of the markup language document.

55. An apparatus for validating a markup language document against a VRD, said apparatus comprising:

means for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

means for storing said first extended hashed representation(s) in a VRD list;

means for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

means for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

56. A computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

code for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

code for storing said first extended hashed representation(s) in a VRD list;

code for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

code for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

57. A computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

code for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

code for storing said first extended hashed representation(s) in a VRD list;

code for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

code for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

58. A method according to claim 7, wherein said code comprises numeric characters.

3. Detailed description of the Invention

Copyright Notice

This patent specification contains material that is subject to copyright protection. The copyright owner has no objection to the reproduction of this patent specification or related materials from associated patent office files for the purposes of review, but otherwise reserves all copyright whatsoever.

Technical Field of the Invention

The present invention relates generally to processing of multimedia documents, and, in particular, to documents produced in mark-up language. The present invention relates to a method and apparatus for parsing documents in mark-up language. The invention also relates to a computer program and a computer program product including a computer readable medium having recorded thereon said computer program, which is configured to make a computer execute a procedure for parsing a document composed in a mark-up language.

Background Art

Parsing is a process of extracting information from a document. The process usually involves at least a minimum check of document syntax, and can in general yield either a tree structure description of the document, or a logical chain of events. The structural representation based on the logical chain of events is typically produced by an ordered parsing of the document from beginning to end.

Tree-based parsers compile, for example, an XML document into an internal tree structure, providing a hierarchical model which applications are able to navigate. The Document Object Model (DOM) working group at the World-Wide Web consortium is presently developing a standard tree-based Application Programming Interface (API) for Extended Markup Language (XML) documents. Event-based parsers, on the other hand, report parsing events such as the start and end of elements directly to the application for which the parsing is being performed. This reporting is performed typically using callbacks, and does not require an internal tree structure. The application requiring the parsing implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based parsers are useful for a wide range of applications, but typically place a strain on system resources, particularly if the document being parsed is large. Furthermore, applications sometimes need to build their own particular tree structures, and it is inefficient to build a tree representation, only to map it to a different representation. Event-based parsers provide a simpler, lower-level access to an XML document, facilitating parsing of documents larger than available system memory. The "Simple API for XML" (referred to as the SAX parser) is an event-driven interface for parsing XML documents. SAX parsers are discussed in more detail in relation to Figs. 2(a), 2(b), 3(a), 3(b) and 3(c).

Figs. 1(a) and 1(b) shows block representations of parser systems. The following XML document fragment 106 is considered:

```

105 <Shakespeare>
110 <!--This is a comment-->
115 <div class="preface" Name1="value1" name2="value2">
120 <mult list=&lt;> </mult>
125 <banquo>
130 Say .
135 <quote>
140 goodnight </quote>,
145 Hamlet.</banquo>
150 <Hamlet><quote>Goodnight, Hamlet. </quote></Hamlet>
155 </Shakespeare>

```

[1]

In Fig. 1(b), the XML document 106 is input into a parser 112 which, in the present instance, is an event based parser. Optionally, as indicated by a dashed box 108, a Document-Type-Definition (DTD) or an XML Schema is also input into the parser 112. The parser 112 outputs, as depicted by an arrow 114, a partial structural representation of the document 106 which can be a simple list. In Fig. 1(a), a Cascading Style Sheet (CSS) or an Extendable Style Sheet (XSL) 104 is input into a CSS or XSL parser 110. A DTD 102 can also be input into this parser 110. Both the XML parser 112 and the CSS/XSL parser 110 are event driven parsers in the present illustration.

One of the benefits of mark-up languages such as XML is the facility to make documents smarter, more portable and more powerful, by enabling the use of tags to define various parts of the documents. This capability derives from the descriptive nature

of XML. XML documents can be customised on a per-subject basis, and accordingly, customised tags can be used to make the documents comprehensible, in terms of the structure, to a human reader. This very attribute, however, often leads to XML documents being verbose and large, and this poses a problem in some instances. For example, where XML documents must be parsed in a hardware-constrained piece of equipment, such as a printer, the typically memory intensive nature of conventional parsing is in conflict with the limited memory which can be accommodated in such equipment. Furthermore, the human readability of XML documents is typically of minimal benefit when the documents are processed by hardware constrained pieces of equipment. Furthermore, tag-string matching operations, which are required to a significant degree in XML document parsing, pose a sometimes unacceptable burden of processing requirements, translating into an unacceptable number of processor cycles. These problems apply to both parser instances shown in Figs. 1(a) and 1(b).

Disclosure of the Invention

It is an object of the present invention to substantially overcome, or at least ameliorate, one or more disadvantages of existing arrangements.

According to a first aspect of the invention, there is provided a method of parsing a markup language document comprising syntactic elements, said method comprising, for one of said syntactic elements, the steps of:

- identifying a type of the element;
- processing the element by determining a hash representation thereof if said type is a first type; and
- augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided a method of validating a markup language document against a VRD, said method comprising steps of:

- (a) processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said processing comprising steps of:

- (i) determining a hierarchy position of said document tag;

(ii) determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(b) processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said processing comprising steps of:

(i) determining a hierarchy position of said tag;

(ii) determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) storing said extended hashed representation of said tag in a list; and

(c) validating said markup language document if said extended hashed representation of said document tag is one of found in said list and is a valid subset of a member of said list.

According to another aspect of the invention there is provided a method of validating a markup language document against a VRD, said method comprising steps of:

(a) processing said VRD, for each structural element identified therein, said processing comprising steps of:

(i) determining syntactic attributes of said structural element;

(ii) determining a hashed representation of said structural element; and

(iii) storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) processing the markup language document, for each document structural element identified therein, said processing comprising steps of:

(i) determining syntactic attributes of said document structural element;

(ii) determining a hashed representation of said document structural element; and

(iii) storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

According to another aspect of the invention there is provided a method of encoding a markup language document comprising syntactic elements, said method comprising, for one of said syntactic elements, steps of:

identifying a type of the syntactic element; and

processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

According to another aspect of the invention there is provided a method of decoding a markup language document comprising encoded syntactic elements, said method comprising, for one of said encoded syntactic elements, steps of:

identifying a type of the encoded syntactic element;

processing the encoded syntactic element by at least one of:

(i) determining an inverse hash representation thereof if said type is a first type; and

(ii) determining a decompressed representation thereof if said type is not a first type; and

(iii) retaining the encoded syntactic element.

According to another aspect of the invention there is provided an apparatus for parsing a markup language document comprising syntactic elements, said apparatus comprising:

identifying means for identifying a type of the element;

processing means for processing the element by determining a hash representation thereof if said type is a first type; and

augmenting means for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided an apparatus for validating a markup language document against a VRD, said apparatus comprising:

(a) means for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said means comprising:

(i) means for determining a hierarchy position of said document tag;

(ii) means for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) means for storing said extended hashed representation of said document tag if said document tag is more deeply nested than an extended hashed representation of a previous document tag;

(b) means for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said means comprising:

(i) means for determining a hierarchy position of said tag;

(ii) means for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) means for storing said extended hashed representation of said tag in a list; and

(c) means for establishing whether said extended hashed representation of said document tag is one of to be found in said list, and is a valid subset of a member of said list, thereby validating said markup language document.

According to another aspect of the invention there is provided an apparatus for validating a markup language document against a VRD, said apparatus comprising:

(a) means for processing said VRD, for each structural element identified therein, said means comprising:

(i) means for determining syntactic attributes of said structural element;

(ii) means for determining a hashed representation of said structural element; and

(iii) means for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) means for processing the markup language document, for each document structural element identified therein, said means comprising:

(i) means for determining syntactic attributes of said document structural element;

(ii) means for determining a hashed representation of said document structural element; and

(iii) means for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) means for comparing syntactic attributes and hashed representations of said each document structural element in the structural representation of the document to corresponding syntactic attributes and hashed representations in said structural representation of said VRD to thereby establish validity of the markup language document.

According to another aspect of the invention there is provided an apparatus for encoding a markup language document comprising syntactic elements, to form an at least partial structural representation of the document, said apparatus comprising:

means for identifying a type of the syntactic element; and

means for processing the syntactic element by one of:

(i) determining a hash representation thereof if said type is a first type;

(ii) determining a compressed representation thereof if said type is not a first type; and

(iii) retaining the syntactic element.

According to another aspect of the invention there is provided an apparatus for decoding a markup language document comprising encoded syntactic elements, said apparatus comprising:

means for identifying a type of the encoded syntactic element;

means for processing the encoded syntactic element by at least one of:

- (i) determining an inverse hash representation thereof if said type is a first type;
- (ii) determining a decompressed representation thereof if said type is not a first type; and
- (iii) retaining the encoded syntactic element.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to parse a markup language document comprising syntactic elements, said program comprising:

- code for identifying a type of an element;
- code for processing the element by determining a hash representation thereof if said type is a first type; and
- code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

- (a) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:
 - (i) code for determining a hierarchy position of said document tag;
 - (ii) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and
 - (iii) code for storing said extended hashed representation of said document tag if said tag is more deeply nested than a previous document tag;
- (b) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said code comprising:
 - (i) code for determining a hierarchy position of said tag;

(ii) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) code for storing said extended hashed representation of said tag in a list; and

(c) code for validating said markup language document if said extended hashed representation of said document tag is one of found in said list, and is a valid subset of a member of said list.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing said VRD, for each structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said structural element;

(ii) code for determining a hashed representation of said structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said document structural element;

(ii) code for determining a hashed representation of said document structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) code for validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to encode a markup language document comprising syntactic elements, said program comprising:

code for identifying a type of the syntactic element; and

code for processing the syntactic element by one of:

- (i) determining a hash representation thereof if said type is a first type;
- (ii) determining a compressed representation thereof if said type is not a first type; and
- (iii) retaining the syntactic element.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to decode a markup language document comprising encoded syntactic elements, said program comprising:

code for identifying a type of the encoded syntactic element;

code for processing the encoded syntactic element by at least one of:

- (i) determining an inverse hash representation thereof if said type is a first type; and
- (ii) determining a decompressed representation thereof if said type is not a first type; and
- (iii) retaining the encoded syntactic element.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to parse a markup language document, said program comprising:

code for identifying a type of the element;

code for processing the element by determining a hash representation thereof if said type is a first type; and

code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a

computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said document tag;

(ii) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(iii) code for storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(b) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said code comprising:

(i) code for determining a hierarchy position of said tag;

(ii) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(iii) code for storing said extended hashed representation of said tag in a list; and

(c) code for validating said markup language document if said extended hashed representation of said document tag is one of found in said list and is a valid subset of a member of said list.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

(a) code for processing said VRD, for each structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said structural element;

(ii) code for determining a hashed representation of said structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(b) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(i) code for determining syntactic attributes of said document structural element;

(ii) code for determining a hashed representation of said document structural element; and

(iii) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(c) code for validating said markup language document if syntactic attributes and hashed representations of said each document structural element in the structural representation of the document conforms to corresponding syntactic attributes and hashed representations in said structural representation of said VRD.

According to another aspect of the invention there is provided an at least partial structural representation a markup language document comprising syntactic elements, said at least partial representation having been produced by a method comprising, for one of said syntactic elements, the steps of:

identifying a type of the element;

processing the element by determining a hash representation thereof if said type is a first type; and

augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided an apparatus for parsing a markup language document comprising syntactic elements, said apparatus comprising:

a processor;

a memory for storing (i) the document, and (ii) a program which is configured to make the processor execute a procedure to parse the document;

said program comprising:

- (i) code for identifying a type of an element;
- (ii) code for processing the element by determining a hash representation thereof if said type is a first type; and
- (iii) code for augmenting an at least partial structural representation of the document using the hash representation if said type is said first type.

According to another aspect of the invention there is provided an apparatus for validating a markup language document comprising syntactic elements against a VRD comprising syntactic elements, said apparatus comprising:

- (a) a processor;
- (b) a memory for storing (i) the document, (ii) said VRD, and (iii) a program which is configured to make the processor execute a procedure to validate the document;

(c) said program comprising:

(ca) code for processing the markup language document, for each document tag identified therein, if said document tag is not a first document tag in a corresponding markup language document tag hierarchy, said code comprising:

(caa) code for determining a hierarchy position of said document tag;

(cab) code for determining an extended hashed representation of said document tag concatenated with a hashed representation of a previous document tag in the document tag hierarchy; and

(cac) code for storing said extended hashed representation of said document tag if said document tag is more deeply nested than a previous document tag;

(cb) code for processing said VRD, for each tag identified therein, if said tag is not a first tag in a corresponding tag hierarchy, said means comprising:

(cba) code for determining a hierarchy position of said tag;

(cbb) code for determining an extended hashed representation of said tag concatenated with a hashed representation of a previous tag in the corresponding tag hierarchy; and

(cbc) code for storing said extended hashed representation of said tag in a list; and

(cc) code for establishing whether said extended hashed representation of said document tag is one of to be found in said list, and is a valid subset of a member of said list, thereby validating said markup language document.

According to another aspect of the invention there is provided an apparatus for validating a markup language document containing syntactic elements against a VRD containing syntactic elements, said apparatus comprising:

(a) a processor;

(b) a memory for storing (i) the document, (ii) said VRD, and (iii) a program which is configured to make the processor execute a procedure to validate the document;

(c) said program comprising:

(ca) code for processing said VRD, for each structural element identified therein, said code comprising:

(caa) code for determining syntactic attributes of said structural element;

(cab) code for determining a hashed representation of said structural element; and

(cac) code for storing said hashed representation and syntactic attributes of said structural element in a structural representation of said VRD; and

(cb) code for processing the markup language document, for each document structural element identified therein, said code comprising:

(caa) code for determining syntactic attributes of said document structural element;

(cab) code for determining a hashed representation of said document structural element; and

(cac) code for storing said hashed representation and syntactic attributes of said document structural element in a structural representation of the document; and

(cc) code for comparing syntactic attributes and hashed representations of said each document structural element in the structural representation of the document to corresponding syntactic attributes and hashed representations in said structural representation of said VRD to thereby establish validity of the markup language document.

According to another aspect of the invention there is provided a method of validating a markup language document against a VRD, said method comprising steps of:

determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

storing said first extended hashed representation(s) in a VRD list;

determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

According to another aspect of the invention there is provided an apparatus for validating a markup language document against a VRD, said apparatus comprising:

means for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

means for storing said first extended hashed representation(s) in a VRD list;

means for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

means for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

According to another aspect of the invention there is provided a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

code for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

code for storing said first extended hashed representation(s) in a VRD list;

code for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

code for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a computer program which is configured to make a computer execute a procedure to validate a markup language document against a VRD, said program comprising:

code for determining first extended hashed representation(s) for most deeply nested syntactic element(s) of a first type in the VRD;

code for storing said first extended hashed representation(s) in a VRD list;

code for determining a second extended hashed representation for a most deeply nested syntactic element of the first type in the markup language document; and

code for declaring said markup language document to not be invalid if said second extended hashed representation is present in the VRD list.

Detailed Description including Best Mode

Where reference is made in any one or more of the accompanying drawings to steps and/or features, which have the same reference numerals, those steps and/or features have for the purposes of this description the same function(s) or operation(s), unless the contrary intention appears.

The inventive concept disclosed in this specification is based on the idea that memory requirements of an XML parser can be reduced, and various performance metrics can be improved, by performing a "perfect" hash of the XML tags, and possibly other elements within an XML file. A hash function is a function, mathematical or otherwise, that takes an input string, and converts it to an output code number called a hash value. A perfect hash function is one which creates a unique code number for a unique input string within a preset domain. The input string can be composed, for example, of alpha numeric

characters, or other characters approved by the World Wide Web Consortium, and must be less than a certain length dictated by the hash process specifics. Alternatively, or in addition, the input string can be constrained in other ways, for example in terms of a probability of code number collision based on input context. This idea allows an arbitrary XML tag to be treated as a numeral or code, which can be stored in numeric form in memory. Since a parser normally preserves some portion of an XML structure in memory as the structure is parsed, conversion of XML tags to unique numerals allows memory requirements to be reduced, and furthermore, allows string-to-string comparisons to be replaced with equivalent, but much faster numerical comparisons.

The principles of the arrangements described herein have general applicability to parsing documents using a wide variety of mark-up languages. For ease of explanation, the disclosed arrangements are described with reference to the XML language. This is not intended, however, to limit the scope of the inventive concept. For example, the disclosed arrangements can also be applied to a UTF-16 transformation format (see International Standard ISO/IEC 10646-1 for further details of UTF-16).

Figs. 2(a) and 2(b) depict a prior art SAX parser process 236, which supports optional well-formedness and/or validation checking sub-processes.

In Fig. 2(a), a mark-up document, in the present case an XML document, is opened in an initial step 200. Thereafter, a decision step 202 tests whether the document contains any unprocessed (ie unparsed) characters, and if this is the case, a character is read and stored in a string in a following step 204. If further characters are, however, not detected in the testing step 202, the parsing process 236 terminates in a step 234.

Following the step 204, a check is performed in a testing step 206 to determine whether a complete syntactic element has yet been assembled, and if so, the parser process 236 proceeds to a "Syntactic Type" identification step 210. If, on the other hand, a complete syntactic element has not yet been assembled, the parser process 236 is directed to a decision block 208 which determines if any further characters are available in the document. If additional characters are available, the parser process 236 is directed according to a "yes" arrow back to the step 204. Alternatively, if no more characters are

available, then the process 236 is directed in accordance with a "no" arrow to the syntactic element type identification step 210.

The "type identification" step 210 identifies a "type" for the assembled syntactic element, after which the element string is placed, in a step 212, into a memory representation of the document structure, thereby augmenting the representation as it has been assembled to this point. The memory representation of the document structure, which is typically, in the case of event driven parsers, a partial structural representation of the document, can be a simple list.

After the step 212, the process 236 is directed to a testing step 242, which determines whether a well-formedness check is to be performed. Well-formedness checks ensure that the document meets appropriate "well-formedness constraints", as defined on page 5 of "Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation, 6 October 2000", which is available on the Internet at <http://www.w3.org/tr/2000/rec-xml-20001006.html>. Well-formedness checks test the document for compliance with general structure rules, particularly whether tags in a document have been properly nested. If such a check is to be performed, then the process 236 is directed in accordance with a "yes" arrow to "a" on a dashed boundary line 246. The dashed boundary line 246, along with reference letters "a" to "d" is mirrored by a corresponding boundary line in Fig. 2(b), in relation to which the process 236 is further described. If the well-formedness check is not to be performed, then the process 236 is directed in accordance with a "no" arrow from the testing step 242 to a testing step 244 which determines whether a "validation check" is to be performed. Validation checks involve a comparison of syntactic elements in a document against validity constraints defined in a Validation Reference Document (referred to as a VRD for the sake of brevity) such as a document type definition (DTD), as described in Section 5.1 of the aforementioned W3C Recommendation. DTDs and XML Schemas are examples of VRDs against which validation checks can be performed, however validation checks as described herein can be performed against other types of VRDs. This comparison procedure verifies correct syntactic placement of elements to a greater extent than the mere well-formedness check. If the validation check is to be performed, then the process

236 is directed in accordance with a "yes" arrow to "b" on the dashed boundary line 246. If, on the other hand, the validation check is not to be performed, then the process 236 is directed in accordance with a "no" arrow to "c" on the dashed boundary line 246.

If the well-formedness check is elected, then the process 236 is directed from "a" on the boundary line 246 to an optional sub-process 238, and in particular to a well-formedness checking step 214 found therein. The optional nature of the process 238 is denoted by the dashed rectangle outline thereof. If the validity check is elected, then the process 236 is directed from "b" on the boundary line 246 to an optional sub-process 240, and in particular to a validity checking step 220 found therein. The optional nature of the process 240 is denoted by the dashed rectangle outline thereof. If the validity check is not elected, then the process 236 is directed from "c" on the boundary line 246 to an action selection step 226.

If the well-formedness check is elected, then after the well-formedness step 214, if an error is detected in the following error checking step 216, corrective action and/or error indication takes place as indicated by an arrow 218. If, on the other hand, no errors are detected, then the parser process 236 is directed from the step 216 to the sub-process 240, in which the validation check is performed in the step 220. As noted, the parsing processing 236 can be directed to the validation checking step 220 either from the error checking step 216, or alternatively, the well-formedness checking sub-process 238 can be by-passed, and the process 236 can be directed directly to the validation checking step 220 from "b" on the boundary line 246. The optional well-formedness sub-process 238 can be bypassed if the appropriate decisions are made in the testing steps 242 and 244 (see Fig. 2(a)).

As noted, the validation checking step 220 involves a comparison of the identified syntactic element in the markup document being considered against a document type definition (DTD). This comparison procedure verifies correct syntactic placement of elements to a greater extent than the mere well-formedness check described in relation to the sub-process 238.

Following the validation step 220, if an error is detected in an error checking step 222, corrective action is taken, and/or an error indication is produced, as depicted by an

arrow 224. Alternatively, if no error is detected, the parser process 236 proceeds to the action selection step 226, where an action is selected based upon the type of the syntactic element being considered. The optional sub-processes 238 and 240 can both be bypassed, if the appropriate decisions are made at the decision steps 242 and 244 in Fig. 2(a). If both of the aforementioned sub-processes are bypassed, then as noted the parsing process 236 is directed from "c" on the boundary line 246 directly to the action selection step 226.

If the syntactic element is a tag, then as depicted by an arrow 228 the tag value, or a representative string, is sent to the application in respect of which the parsing process is being performed, and a memory representation of the tag is maintained. If, on the other hand, the element type is a non-tag type, then as depicted by an arrow 230, the element value string is sent to the associated application, and the memory representation of the element is deleted. Finally, the parsing process 236 is directed, as depicted by an arrow 232, to "d" on the dashed boundary line in Fig. 2(b), and from "d" on the corresponding dashed boundary line 246 in Fig. 2(a) to the character testing step 202.

Significant memory requirements arise from the verbose nature of the XML document, resulting in correspondingly significant memory requirements to store the document structure in its original string form. This document structure is referred to in the step 212. Furthermore, an associated significant processing load, relating to performance of string comparisons between variable length alpha-numeric strings, arises both in the well-formedness checking step 214, and in the validation checking step 220.

A partial memory representation of the document must typically be stored, and string checking must typically be performed, both (i) in relation to the step 214 in regard to checking for closure of hierarchy branches, namely matching end tags to start tags, and also for checking for non-overlapping branches, and (ii) in relation to the step 220, in which similar processes are required as in (i), as well as checking conformity of structure and tag names against the DTD.

A parser must normally preserve some portion of an XML structure in memory as the XML structure is parsed. Even for a SAX parser, a local portion of the XML structure must be retained in memory for correct operation. If however each XML tag is converted to a unique numeral using a hash function, memory requirements are typically

reduced, since the numeral resulting from the hash operation is smaller than the associated arbitrary-length XML tag string. Furthermore, string-to-string comparisons, required for matching beginning & end tags, can be replaced with much faster numerical comparisons, thereby reducing the processing load.

Typical hash algorithms include (i) Cyclic Redundancy Coding (CRC) algorithms (commonly used for signature analysis or error-detection/correction in data transfer & storage), (ii) fully lossless encoding algorithms, and (iii) Huffman encoding algorithms.

Typically, a suitable hash algorithm must be static in its operation, or in other words it must always return the same hash result for the identical input conditions over the required set of data. The required set of data can, however, vary according to the circumstance. The data set can thus typically comprise at least an entire markup document, but can also include a relevant DTD or XML Schema, linked markup documents, and related or linked markup documents in different languages, eg a CSS document referenced by an XML document. A static hash algorithm can, however, be used where necessary by resetting the algorithm whenever tag syntax is encountered, for example whenever the non-literal '<' character is found in an XML document. The hash algorithm can also be reset where an <IELEMENT string is found in an XML DTD document, or wherever a valid tag selector is permitted in a CSS document.

A reference in an input markup document can be used to signal, or to select a suitable hash algorithm. This can be done in much the same way as markup documents can reference other markup documents, DTDs, stylesheets, character encodings, namespaces, and so on. For example a particular hash algorithm can be identified with a particular namespace, thereby permitting indirect reference to a hash algorithm via a namespace reference within a document. A hash algorithm implementation can be wholly, or partially included within a markup document, along with associated parameterisation. Such methods of referencing or including hash algorithms can be useful for optimisation purposes, where different hash methods have been optimised for use with particular markup documents, thereby improving performance and memory usage in destination devices or systems. Alternatively, the aforementioned referencing methods can be useful

for matching purposes. This refers to applications involving one or more markup documents, where error checking or completion of parsing or other functions are required, and where one or more other documents (e.g. a DTD) have already been hashed by the same algorithm.

Further refinements are possible in the above approach, for example involving optional hashing of DTDs. This reduces Read Only Memory (ROM) requirements for storing DTDs, and provides for faster validation processing of XML documents, by allowing comparison of numerical values rather than (slower) string comparisons.

Figs. 3(a), 3(b) and 3(c) illustrate one arrangement of an improved SAX parser process 344. In Fig. 3(a), steps 300, to 310 are identical to corresponding steps 200 to 210 which have been described in relation to Fig. 2(a). After the step 310, the assembled syntactic element is tested to ascertain its nature as a tag, or another element type, in a testing step 312. If the element is a tag, the parsing process 344 is directed to a hash step 318 by an arrow 316. The hash step 318 determines, using respective processors 414 or 505 in Figs. 5 and 6, a unique numeric representation of the syntactic element. This results in a more memory efficient representation of the element, which also lends itself to simpler and faster comparison operations in the numeric, rather than the alpha-numeric domain. Both the element string depicting the syntactic element, and the hash value thereof, are retained at this point of the process 344, however it is the hash value, and not the string value, which is inserted, in the step 318, into the memory representation of the document structure using respective memories 418 and 506 (see Figs. 5 and 6).

In order to better appreciate operation of the parsing process 344 as described in relation to Figs. 3(a), 3(b) and 3(c), parsing of the exemplary XML fragment [1] is considered firstly in relation to the parsing process 236 described in relation to Fig. 2. In this case, the XML fragment [1] yields the following hierarchical representation of parsed mark-up tags in the sub-process 212:

```

205 Shakespeare
215   div
220     mult
221   /mult
225   banquo
235     quote
240   quote
245   /banquo
250   Hamlet
251     quote
252   /quote
253   /Hamlet
255 /Shakespeare

```

[2]

In contrast, the differentiated treatment of tag elements and non-tag elements in the parsing process 344, as described in relation to Fig. 3(a), results in an equivalent hierarchical representation being generated by the step 318. The equivalent hierarchical representation is depicted in [3]. The hierarchical representation in [3] is made up of parsed hashed mark-up tags. For the sake of this example, a domain of tag names is constrained to those shown in the following Table 1, and a hash mapping (which is functionally equivalent to application of a hash "function") is shown in the following table:

Tag	Hash Code Number
Shakespeare	133
Div	326
Mult	371
Banquo	787
Quote	629
Hamlet	411

Table 1. Hash Mapping

Based on the above hash mapping, the following hierarchical representation of the XML fragment shown in [1] results:

205	133			
215		326		
222		371		
223		/371		
225		787		
235			629	
240			/629	
245		/787		[3]
254		411		
255			629	
256			/629	
257		/411		
255	/133			

Returning to Fig. 3(a), the parsing process 344 is directed from the step 314 to "a" on a dashed boundary line 356. The dashed boundary line 356, along with reference letters "a" and "b" is mirrored by a corresponding boundary line in Fig. 3(b), in relation to which the process 344 is further described.

Turning to Fig. 3(b), the process 344 continues from "a" on the dashed boundary line 356 to a testing step 350 which determines whether a well-formedness check is to be performed. If such a check is to be performed, then the process 344 is directed in accordance with a "yes" arrow to "c" on the boundary line 358. The dashed boundary line 358, along with reference letters "c" to "f" is mirrored by a corresponding boundary line in Fig. 3(c), in relation to which the process 344 is further described. If the well-formedness check is not to be performed, then the process 344 is directed in accordance with a "no" arrow to a testing step 352 which determines whether a validation check is to be performed. If the validation check is to be performed, then the process 344 is directed in accordance with a "yes" arrow to "e" on the dashed boundary line 358. If, on the other hand, the validation check is not to be performed, then the process 344 is directed to "d" on the dashed boundary line 358.

Turning to Fig. 3(c) if the well-formedness check is to be performed, then the process 344 is directed from "c" on the dashed boundary line 358 to a well-formedness checking step 320. If, on the other hand, the well-formedness check is not elected, then the process 344 is directed from "e" on the dashed boundary line 358 to a validation checking step 326. If neither a well-formedness check or validation check is elected, then

the process 344 is directed from "d" on the dashed boundary line 358 to an action selection step 334.

The well-formedness checking step 320 performs well-formedness checking using respective processors 414 or 505 and forms part of an optional process 346. The optional nature of the process 346 is depicted by use of dashed lines. Similarly, the validation step 326 forms part of an optional sub-process 348, the optional nature thereof being depicted by use of dashed lines.

It is apparent that the hierarchical representation depicted in [3] allows string comparisons to be replaced by faster and more efficient numerical comparisons, thereby reducing the associated computational burden. Furthermore, the hierarchical representation shown in [3] is a more memory-efficient representation, than that shown in [1] and accordingly the representation shown in [3] is more suited to memory-constrained applications as previously discussed.

Returning to Fig. 3(c), if well-formedness checking is elected, then after well-formedness checking is performed in the step 320, the parsing process 344 is directed to an error checking step 322, whereupon if an error is detected, as depicted by an arrow 324, corrective action is taken, and/or an error is indicated. The well-formedness check typically considers whether tags in a document have been properly nested. Thus, for example, having reference to [2] the tag pair "Hamlet" and "/Hamlet" are properly nested within the tag pair "Shakespeare" and "/Shakespeare" since the "Hamlet" tag pair is fully nested within the "Shakespeare" tag pair, and the tag pairs do not, for example, overlap each other.

If, on the other hand, no error is detected, the parsing process 344 is directed to the optional process 348, in which the validation checking step 326, using respective processors 414 or 505, is performed with reference to a DTD or an XML Schema. As noted, validation checking is a more detailed form of checking than well-formedness checking. Thus, for example, whereas the well-formedness check considers whether the "Hamlet" tag pair is properly nested within the "Shakespeare" tag pair, validity checking, in contrast, both checks for proper nesting in the sense that the "Hamlet" tag pair is fully nested within the "Shakespeare" tag pair, but also checks whether "Hamlet" tag pairs may

legally be nested in this way. There may, for example, be a situation where, in fact, "Shakespeare" tag pairs must be nested within "Hamlet" tag pairs, rather than the other way around. Thus, the validity checking process checks hierarchical relationships of tags, in this case being whether "Hamlet" tag pairs may be nested within "Shakespeare" tag pairs, as well as considering whether nesting has been properly, namely completely, performed.

In order to perform the validation step 326, DTD or XML Schema tags are first hashed in a hashing step 328, in order to bring the DTD/XML Schema memory representation into conformity with the hashed nature of the mark-up document which has been generated by the hash step 318. The validation checking step 326 compares the mark-up document structural representation generated in the step 318 to the structural representation of the DTD/XML Schema generated in the step 328, to verify correct syntactic placement of syntactic elements in the markup document, noting that the string comparisons required for this comparison as used in step 220 in relation to Fig. 2, are now replaced, in Fig. 3(c), by faster and more efficient numerical comparisons, as a result of the hashing operations in steps 328 and 318.

After validation, the process 344 is directed to an error checking step 330, in which corrective action and/or error indication is performed as indicated by an arrow 332. If no errors are detected, the parsing process 344 proceeds to an action selection step 334, whereupon if the syntactic element is a tag type, the corresponding tag string is sent to the application in respect of which the parsing process is being performed, and the tag string itself is deleted from memory, this being either 418 or 512 in Figs. 5 and 6 respectively. The associated hashed tag memory representation is, however, retained. Accordingly, no string-based memory representation of the tag is retained, other than one copy of the currently parsed tag string. The memory representation of the tag is thus only in hashed form. If the element syntactic type is either a non-tag type, or a non-tag name type, then as depicted by an arrow 338, the value of the element, or a string representation thereof is sent to the associated application, and the associated memory representation is deleted. The parsing process 344 now loops back, as indicated by an arrow 340, to "f" on the dashed boundary line 358, and thereafter to the corresponding "f" on the dashed boundary

line 358 in Fig. 3(b), and thereafter to "b" on the dashed boundary line 356, and thereafter to "b" on the dashed boundary line 356 in Fig. 3(a), and finally to the character testing step 302. If no further characters are detected, the parsing process 344 terminates in a step 342.

The XML document fragment [1], with tags in hashed form, has the following form:

```

505 <133>
110 <!--This is a comment-->
515 <326 class="preface" Name1="value1" name2="value2">
520 <371 list=&lt;> </371>
525 <787>
130 Say
535 <629>
540 goodnight </629>,
545 Hamlet.</787>
550 <411><629>Goodnight, Hamlet. </629></411>
555 </133>

```

[4]

The representation of closing tags (which typically use syntax: </section> as opposed to start tags which use syntax <section>) can be defined in various ways, thereby attaining more, or less, compatibility with the XML standard. It is noted that start tags and end tags are considered, in the present description, to be "equivalent types". Furthermore, the fact that the start and end tag perform a collective function, namely delimiting sections of document content, is taken to mean that there is a relationship between the two tags. It is further noted that the aforementioned syntax for start and end tags means that the end tag is a modification of the start tag, wherein a distinguishing character, namely a "/" is incorporated into the start tag in order to produce a corresponding end tag. Compatibility with the XML standard can be more important in some instances than in others. In the preferred embodiment, the '/' character of a current tag string is typically removed prior to hashing the following tag name, in order that identical start and end tag names return the same numeral from the hashing function. An XML tag is exemplified by </Name Attribute>, as can be seen in Section 3.1 of "Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation, 6 October 2000", which is available on the Internet at <http://www.w3.org/tr/2000/rec-xml->

20001006.html). In the present description, the term "tag" can refer, depending on the context, to either a part of, or the entirety of, the particular tag being considered. Alternatively, there may be situations where it is desired to retain an equivalent representation of the '/' character (identifying the end tag) in memory. This can be done in a variety of ways, such as: (i) reinserting the '/' character or an equivalent character into memory in proximity to the end tag hash numeral so as to indicate that it is an end tag, (ii) using a boolean value to indicate the end or start state of a hashed tag, or (iii) negating the end tag hashed value so that a simple addition of start and end tags yields zero for a perfect match. In Option (iii), the hashed start tag has been modified by an operator, in the present case a simple negation operation, in order to produce the requisite hashed end tag. Option (iii) requires that a sign bit be guaranteed to be free from influence by the hashing algorithm. This option is, in fact, very similar to the boolean flag option (ii).

Furthermore, structured hash numbers can be generated in which a hash number for a nested tag can explicitly indicate the higher-level XML tags within which the first tag is nested. Thus, for example, where tag 123 is nested inside tag 987, then instead of being designated as nested tag 123, it can be designated as 987.123. This structured, or "extended", hashing can allow further parsing performance improvements by reducing structure-spanning operations, ie by reducing an amount of the XML document which must be held in memory while the end-points of a tag pair are being searched for.

It is also noted that extended representations need not be based upon hashing, but can also be based upon strings, or "enumeration", which is a process whereby a mapping is defined between tag names and numerals, thereby creating an enumeration table or index. A simple form of enumeration is to merely list all the tag names, and to number the listed tags. Thus, for example, a concatenated string of the form "Shakespeare.banquo.quote" represents a string-based extended representation of three concatenated tags.

A structured equivalent hashed markup example for the XML fragment [3] is presented in [5] below using negated, hashed end tags.

```

133
133.326
133.371
133.-371
133.787
133.787.629      -> 013307870629
133.787.-629     -> -013307870629
133.-787
133.411
133.411.629
133.411.-629
133.-411
-133

```

[5]

In [5], the structure of nested tags is converted from the form shown in [3] into a series of concatenated hashed tags, in which each subsequent lower (ie. more deeply nested) hierarchical level of hashed tag is directly linked to its previous upper hierarchical levels. This allows simple numerical comparison to be performed with a similarly parsed structure from a hashed DTD. In fact, each line in [5] is represented, as shown in [5] for lines 4 and 5, by a single numeral which is combined by concatenation of the set of hashed tags encountered. This single numeral represents in a very compressed form both the identities and relationships of the original input tags, and accordingly enables a very efficient comparison method with a similarly hashed DTD. It can be seen that the numerical tag sets can be used to represent the document structure in a highly compressed form. A validation check can be performed using merely the hashed start tag sets, noting that each such set represents the deepest, and entire, structure of each branch of the document structure. For instance, the structure of [5] can be minimally represented in [6] as follows:

```

01330326
01330371
013307870629
013304110629

```

[6]

A DTD or XML schema structure can also be represented by the same method.

A single, or multiple set of numerical comparisons between a tag set from the parsed & hashed input document and a tag set from the parsed & hashed DTD replaces a series of string and structure comparisons normally required in XML parser validation. It

can be recognized that any alternate valid structures defined by a DTD or XML schema can be encoded into unique hashed tag set numerals for later comparison with hashed tag set numerals generated from an input XML document.

Fig. 4 depicts a process 600 for validating a mark-up document against a VRD such as, for example, a DTD, or an XML schema. The process 600 commences with a step 602 in which a markup document to be validated is opened. Thereafter, in a step 604, a current extended tag is reset by a respective processor 414 or 505 in Fig. 5 or Fig. 6. In the description relating to Fig. 4, the terms "tag", "extended tag", "temporary tag" and so on refer to the hashed representations of the respective tags. In a following step 606, a temporary tag root is reset by one of the respective processors 414 or 505, after which a next tag in the markup document is identified in a step 608. Thereafter, a testing step 610 determines whether the tag identified in the step 608 is a start tag, in which event the process 600 is directed in accordance with a "yes" arrow to a step 612 which adds the tag identified in the step 608 to the extended tag using one of the respective processors 414 or 505. The process 600 is then directed from the step 612 back to the step 608.

If the testing step 610 determines that the next tag is not a start tag, then the process 600 is directed in accordance with a "no" arrow to a testing step 614, which determines whether the extended tag = "0", which represents the root level of the document. If the "0" value is detected, then the process 600 is directed in accordance with a "yes" arrow to a testing step 624 which determines whether the end of the document has been reached. If this is not the case, then the process is directed in accordance with a "no" arrow back to the step 606. It is noted that detection of a "0" value in the step 614 may also result from an document structure which is not well formed, such as would be the case for a structure having a mismatched number of start and end tags.

If the testing step 614 determines that the extended tag value is not equal to "0", then the process is directed in accordance with a "no" arrow to a testing step 616, which determines whether the extended tag is equal to the temp tag root value. If this is found not to be the case, then the process 600 is directed in accordance with a "no" arrow to a step 618 which stores the extended tag in a document list in a respective memory 418 or

506. If on the other hand, the testing step 616 determines that the extended tag is equal to the temp tag root, then the process 600 is directed in accordance with a "yes" arrow to a step 620 which removes the lowest (namely the most deeply nested) tag from the extended tag, using a respective processor 414 or 505. Thereafter, in a step 622, the process 600 copies the extended tag to the temp tag root, after which the process is directed back to the step 608.

Prior to returning to the testing step 624, it is noted that the process 600 as heretofore described is directed to the markup document whose validity is being checked. There is also, however, an identical process, not explicitly described, which is applied to the validation reference document (VRD) to thereby produce a VRD list against which the document list produced by the process 600 can be tested. The process 600 and the equivalent process directed to the VRD typically occur at different times. The process 600 occurs for every document being validated, and produces a list of extended hash representations for each particular document being validated. The VRD list can be produced substantially concurrently with the process 600, providing that the VRD list is completed prior to the step 626. Alternately, the VRD process can be performed off line, and the resultant list provided to the process 600 prior to the step 626.

Returning to the step 626, and since the VRD list is available as noted, the step 626 determines whether every entry in the document list is to be found in the VRD list. If this is the case, then the process 600 is directed in accordance with a "yes" arrow to a step 628 which declares that the document is not detected as invalid. If on the other hand, the document list has an entry which is not to be found in the VRD list, then the process 600 is directed in accordance with a "no" arrow to a step 630 which declares that the document is invalid.

The above description compares, as described in more detail in regard to the step 626, all the document list entries with all the VRD list entries. An alternate process is to test each extended tag, after the step 616, against the complete VRD list in a step similar to the step 626, in which event if the document extended tag is not to be found in the VRD list, the process 600 can proceed directly to the step 630, saving unnecessary further testing. In the event that the extended tag is however to be found, then the process 600

can be directed to the step 620 and so on. The alternate arrangement provides earlier recognition of an error, and immediately aborts the validation process, which provides added efficiency provided the VRD list is relatively short. If a complete validation check is implemented with the above method, then the step 628 indicates that the document being considered is valid.

In order to further illustrate the validation method described, the following structure fragment is considered, in which start tags are "01" to "05", and the corresponding end tags are "-01" to "-05" respectively.

```

01
  02
    03
    -03
    04
    -04
  -02
  05
  -05
-01

```

[7]

In functional terms, the process 600 traverses down to the deepest part of a branch in the hierarchical structure of the mark-up document, namely from "01" on the first line of [7] to "03" on the third line of [7], and stores an extended hash representation for the deepest part of that particular branch, namely "010203". The process then traverses up the branch, discarding end tags, until it finds another start tag which indicates a new branch to pursue, which is "04" on line 5 of [7] in this example. As the process traverses down the new branch, the process preserves the extended hash representations of higher levels of the hierarchy, until it has stepped back above those levels. An error in the document structure, resulting in an invalid document or a document which is not well formed, will typically return extended hash representations that do not match those of the VRD. The step 620 may optionally include well-formedness checks of the retrieved end tag against the previous start tag, thereby providing a well-formedness match if the document is well formed. It is noted that the previous start tag is the lowest tag in the extended hash representation. For example the DTD/XML Schema may return end tags

at the testing step 610 that do not match the lowest start tag in the extended hashed representation in the step 620, thereby failing a well-formedness check.

The test at the step 626 typically seeks to match the extended hashed representations of the mark-up document structure against those hashed representations listed for the DTD of that document. The hashed representations of the document structure will typically be a subset of the deepest structural representations from the DTD list. Accordingly, a valid XML document is permitted to contain any legal subset of the structural nesting defined in the corresponding VRD, or DTD. Therefore, a typical test in the step 626 includes comparisons of shallower hashed structural representations of the document against a deeper hashed representation of the DTD. Thus, for example, an extended hashed representation "0123" from the XML document would be assessed as "valid" when compared to a hashed representation "01230456" from the corresponding DTD.

The validation process 600 shown in Fig. 4 can be optimized to check the more complex parts of a document structure, such as the most deeply nested portions, in a fast but incomplete check of validity. Thus, an optimal combination of speed and validity sensitivity can be selected, in order to implement a particular validating parser having arbitrary performance characteristics.

The validation method 600 can also be modified to perform at least portions of "standard" well-formedness checks. Thus, for example, in the step 620, the hashed representation of the end tag can be checked against the lowest hierarchical hashed tag representation within the extended tag representation. If the aforementioned representations do not properly resolve to the same original tag identity, then the document is not well-formed, and a recovery, or error action can be performed.

The above method can be extended to include hashed representations of defined attributes within a structure, either separately, or together with structure checking.

It is apparent that this method of validation and well-formedness checking can be applied to an input document in a separate process to the process for parsing of the document structure and content. Thus, for example, the method 600 can be optimised in order to achieve an efficient and high-speed validity and well-formedness check that can

be performed even in environments where central processing unit (CPU) cycles and memory size are not particularly subject to major constraints. The advantage of performing a separate check in such systems includes the fact that a highly optimized check can be used to quickly discard "invalid" documents. This can save considerable time and processing of at least part of an invalid document, thereby preventing, for example, (i) parsing of the document into a full DOM tree representation and then performing validation checking only to find that the document is invalid, or (ii) commencing further processing of a first (valid) part of a document prior to detecting an invalid second part of the document, the further processing of the first part of the document being thereby rendered futile. Another advantage is that after a document is discovered to be invalid using the fast validation check, processing of a following job can be immediately commenced.

An "imperfect" hash process ie a hash process which is not guaranteed to produce a unique numeral for each alphanumeric input string, can be adequate in certain cases, in particular where the maximum length of XML tag strings is constrained, or is at least constrained to some level of probability. Furthermore, in cases where the set of XML tag strings is constrained to some limited number of character permutations, or is constrained with some probability to a limited number of character permutations, the imperfect hash process can be designed, or selected, to operate adequately.

A communications standard, or alternative public or private format(s) for numerical representation of a document structure can be defined or described based on the use of a hash algorithm. This technique allows a form of compression, which can be of benefit in transmission of XML data which normally involves transmission of a significant amount of data because of its verbosity, and human-readable ASCII form. Various options exist for retaining or discarding human-readability, for example by combining (perfect) hashing with other forms of compression, which are respectively applied to differing element types within an XML file. For instance, it is possible to replace XML string tags with unique, human-readable numerals derived from a perfect hashing algorithm. Un-hashed syntactic and other elements can also be compressed by a

lossless compression technique for transmission between processes or devices, thereby reducing the amount of transmitted data.

An inverse or reversible hash algorithm can be referenced or included where required as discussed in the previous paragraph. This is used where, for example, such an algorithm is needed to decode or decrypt one or more markup tags into a human-readable string for display or labelling purposes from a pre-hashed, transmitted markup document, where it is otherwise not necessary to do so for parsing and error-checking purposes. Another use of a reverse or inverse hash algorithm is to allow decryption of markup tags or other data to enable a restricted function or feature relating to the transmitted markup document. Reverse or inverse algorithms can also be used for matching a transmitter and a receiver of markup documents, where the reverse or inverse hash algorithm is already included in the receiver, and is not transmitted, but might be referenced in the markup document. Examples of reversible or invertible hash algorithms include (i) fully lossless encoding algorithms and (ii) Huffman encoding algorithms.

The aforementioned arrangements can be applied to any markup language, with particular advantages where one or more of the following conditions apply, namely (i) the markup language allows definition of tag names (e.g. XML, DTD, CSS, XSL, etc), (ii) tag names use large character encoding tables (e.g. UTF-16) and/or tag name length is not typically shorter than the hashed representation thereof, (iii) the intended application using or receiving a markup document typically requires representation of complex structures with more than one hierarchical level of nesting within a markup document, XML Schema, or DTD, (iv) some form of checking, typically well-formedness or validation, is required for the input markup document, (v) the markup parser and/or application have strong limitations on memory capacity (for example, embedded or low-cost CPU systems) or memory management (for example in systems having no virtual memory, or no dynamic memory allocation), and (vi) the markup parser and/or application need to operate quickly on potentially complex, highly-nested, markup documents.

The disclosed method of parsing a markup language document can be implemented in dedicated hardware such as one or more integrated circuits performing

the functions or sub functions of parsing a markup language document. Such dedicated hardware may include graphic processors, digital signal processors, or one or more microprocessors and associated memories.

The method of parsing a markup language document can alternatively be practiced using a special purpose embedded computer system 400, such as that shown in Fig. 5 wherein the processes of Figs. 3(a), 3(b), 3(c), and 4 may be implemented as software, such as an application program executing within the embedded computer system 400. The computer system 400 is typically integrated (embedded) into an end system such as a printer (not shown) and drives a printer engine 402 in the printer. In particular, the steps of the method of parsing a markup language are effected by instructions in the software that are carried out by the embedded computer. The software may be stored in a computer readable medium, including Read Only Memory (ROM) 418 or Random Access Memory (RAM) 418 or other types of memory (not shown). The software is loaded into the embedded computer during manufacture, or by software upgrades performed on-site.

The embedded computer system 400 comprises a computer module 410, input devices such as a switch module 422 for parameter setting, an output device such as a Liquid Crystal Display (LCD) showing job status, and the printer engine 402. The embedded computer 400 is typically physically integrated into the printer (not shown). Print jobs which originate at other computers (not shown) attached to a computer network 406 are sent to the embedded computer 400 by a connection 404 to an Input/Output (I/O) interface 408.

The embedded computer module 410 typically includes a processor unit 414, a memory unit 418, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), input/output (I/O) interfaces including a switch module and LCD interface 416, and an I/O interface 408 for the printer engine 402 and network 406. The components 408, and 414 to 418 of the embedded computer 410 typically communicate via an interconnected bus 412 and in a manner which results in a conventional mode of operation of the embedded computer system 410 known to those in

the relevant art. Typically, the program of the arrangement is resident in memory 418, and is read and controlled in its execution by the processor 414.

The method of parsing a markup language document can also be practiced using a conventional general-purpose computer system 500, such as that shown in Fig. 6 wherein the processes of Figs. 3(a), 3(b), 3(c), and 4 may be implemented as software, such as an application program executing within the computer system 500. This application is useful, for example, when hashing is used as a communication standard across a network between computers. Fig. 6 shows only one of the communicating computers being considered.

In particular, the steps of the method of parsing a markup language document are effected by instructions in the software that are carried out by the computer. The software may be divided into two separate parts, namely one part for carrying out the parsing methods, and another part to manage the user interface between the latter and the user. The software may be stored in a computer readable medium, including the storage devices described below, for example. The software is loaded into the computer from the computer readable medium, and then executed by the computer. A computer readable medium having such software or computer program recorded on it is a computer program product. The use of the computer program product in the computer preferably effects an advantageous apparatus for parsing a markup language document in accordance with the embodiments of the invention.

The computer system 500 comprises a computer module 501, input devices such as a keyboard 502 and mouse 503, output devices including a printer 515 and a display device 514. A Modulator-Demodulator (Modem) transceiver device 516 is used by the computer module 501 for communicating to and from a communications network 520, for example connectable via a telephone line 521 or other functional medium. The modem 516 can be used to obtain access to the Internet, other network systems, such as a Local Area Network (LAN) or a Wide Area Network (WAN), and the other personal computer (PC) 522 with which the computer 500 is communicating.

The computer module 501 typically includes at least one processor unit 505, a memory unit 506, for example formed from semiconductor random access memory

(RAM) and read only memory (ROM), input/output (I/O) interfaces including a video interface 507, and an I/O interface 513 for the keyboard 502 and mouse 503 and optionally a joystick (not illustrated), and an interface 508 for the modem 516.

A storage device 509 is provided and typically includes a hard disk drive 510 and a floppy disk drive 511. A magnetic tape drive (not illustrated) may also be used. A CD-ROM drive 512 is typically provided as a non-volatile source of data. The components 505 to 513 of the computer module 501, typically communicate via an interconnected bus 504 and in a manner which results in a conventional mode of operation of the computer system 500 known to those in the relevant art. Examples of computers on which the embodiments can be practised include IBM-PC's and compatibles, Sun Sparcstations or alike computer systems evolved therefrom.

Typically, the application program of the embodiment is resident on the hard disk drive 510, and is read and controlled in its execution by the processor 505. Intermediate storage of the program and any data fetched from the network 520 may be accomplished using the semiconductor memory 506, possibly in concert with the hard disk drive 510. In some instances, the application program may be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive 512 or 511, or alternatively may be read by the user from the PC 522 over the network 520 via the modem device 516.

Still further, the software can also be loaded into the computer system 500 from other computer readable medium including magnetic tape, a ROM or integrated circuit, a magneto-optical disk, a radio or infra-red transmission channel between the computer module 501 and another device, a computer readable card such as a PCMCIA card, and the Internet and Intranets including email transmissions and information recorded on websites and the like. The foregoing is merely exemplary of relevant computer readable mediums. Other computer readable mediums may be practiced without departing from the scope and spirit of the invention.

Industrial Applicability

It is apparent from the above that the embodiment(s) of the invention are applicable to the computer and data processing industries.

The foregoing describes only some embodiments of the present invention, and modifications and/or changes can be made thereto without departing from the scope and spirit of the invention, the embodiments being illustrative and not restrictive.

Brief Description of the Drawings

A number of preferred embodiments of the present invention will now be described with reference to the drawings, in which:

Figs. 1(a) and 1(b) shows block representations of XML parser systems in which embodiments of the present invention can be practiced;

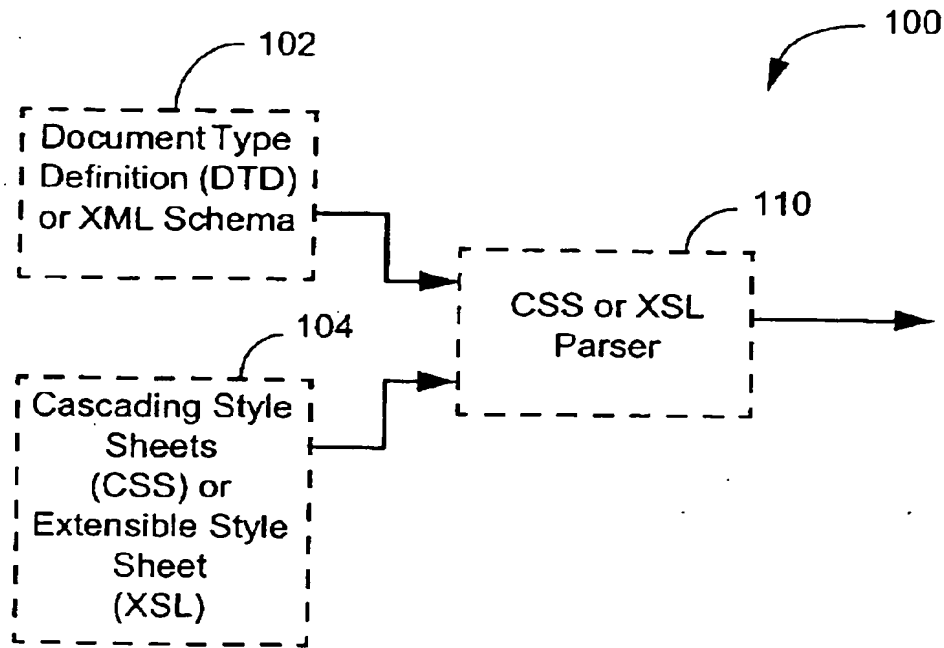
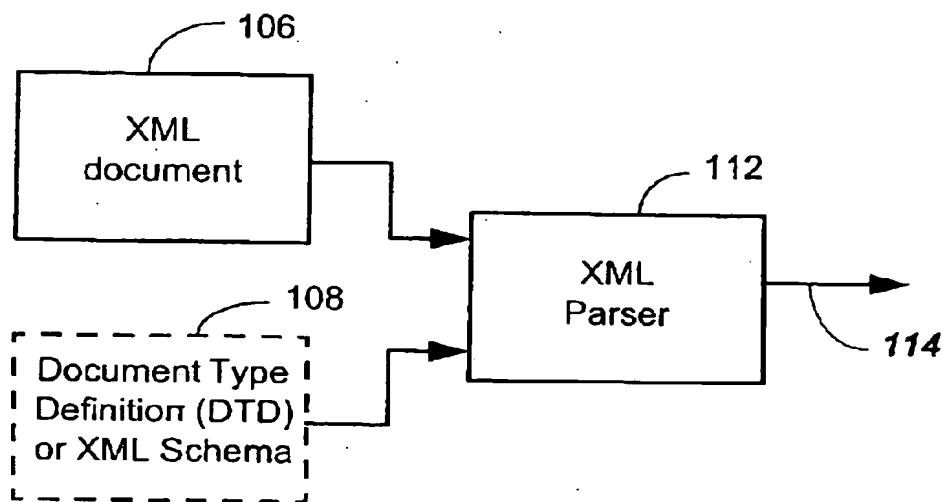
Figs. 2(a) and 2(b) depict a flow chart of method steps for a prior art SAX parser, including optional well-formedness and/or validation checking steps;

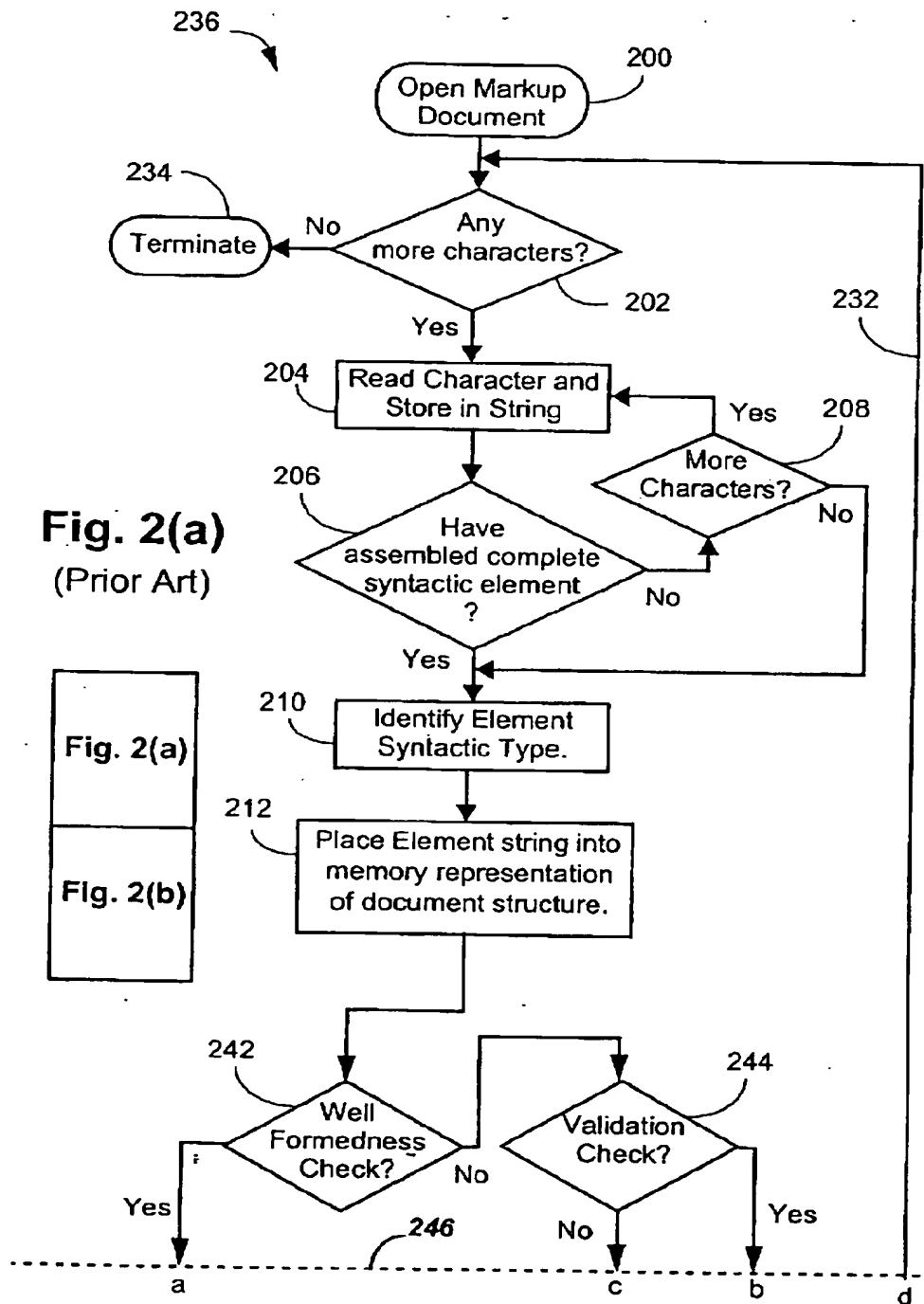
Figs. 3(a), 3(b) and 3(c) show an improved arrangement of the SAX parser of Figs. 2(a) and 2(b);

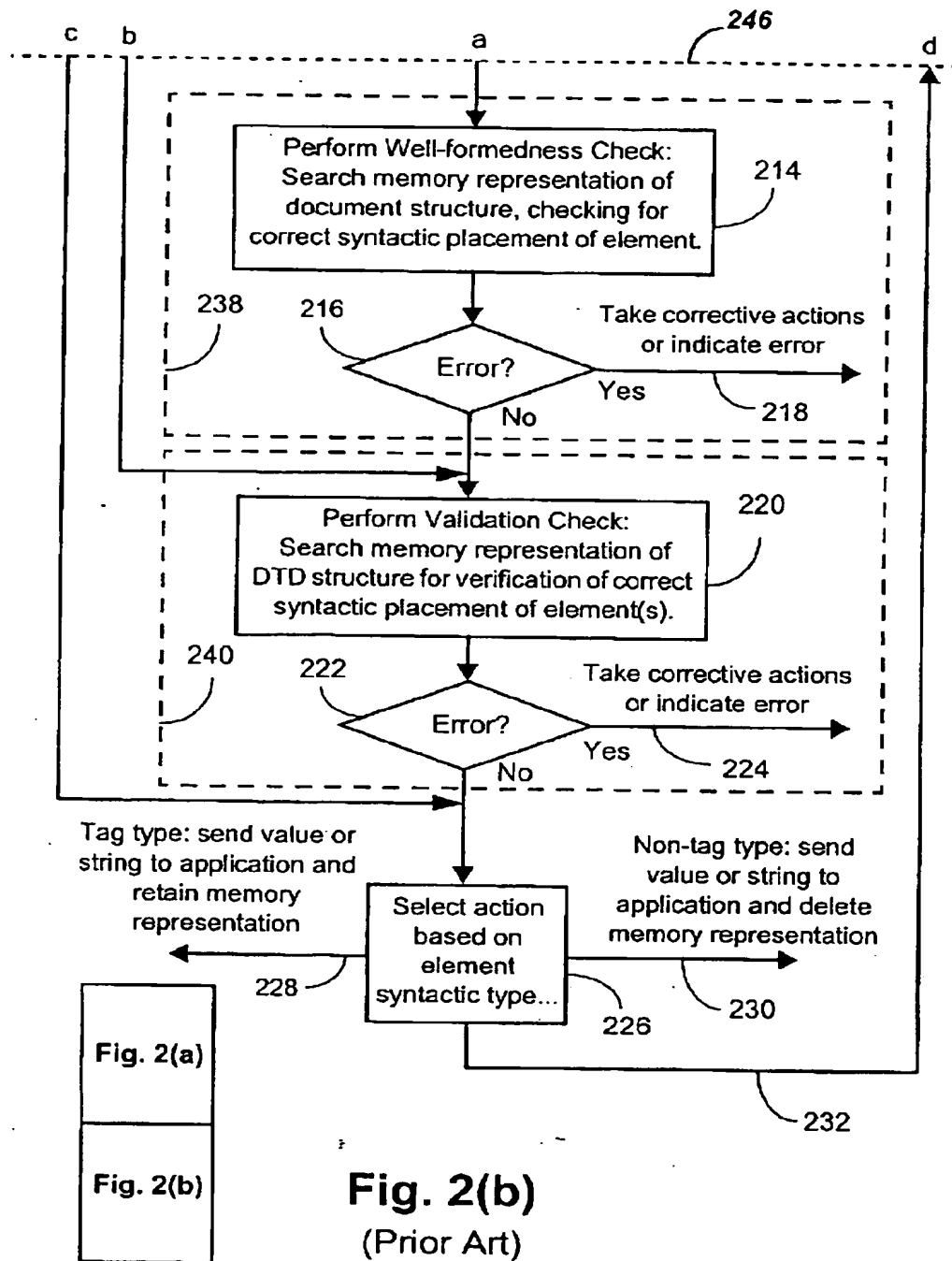
Fig. 4 depicts a process for validating a document against a reference document such as a DTD, or an XML schema.

Fig. 5 is a schematic block diagram of a special purpose embedded computer upon which an arrangement of the improved SAX parser can be practiced; and

Fig. 6 is a general purpose computer upon which an arrangement of the improved SAX parser can be practiced.

**Fig. 1(a)****Fig. 1(b)**





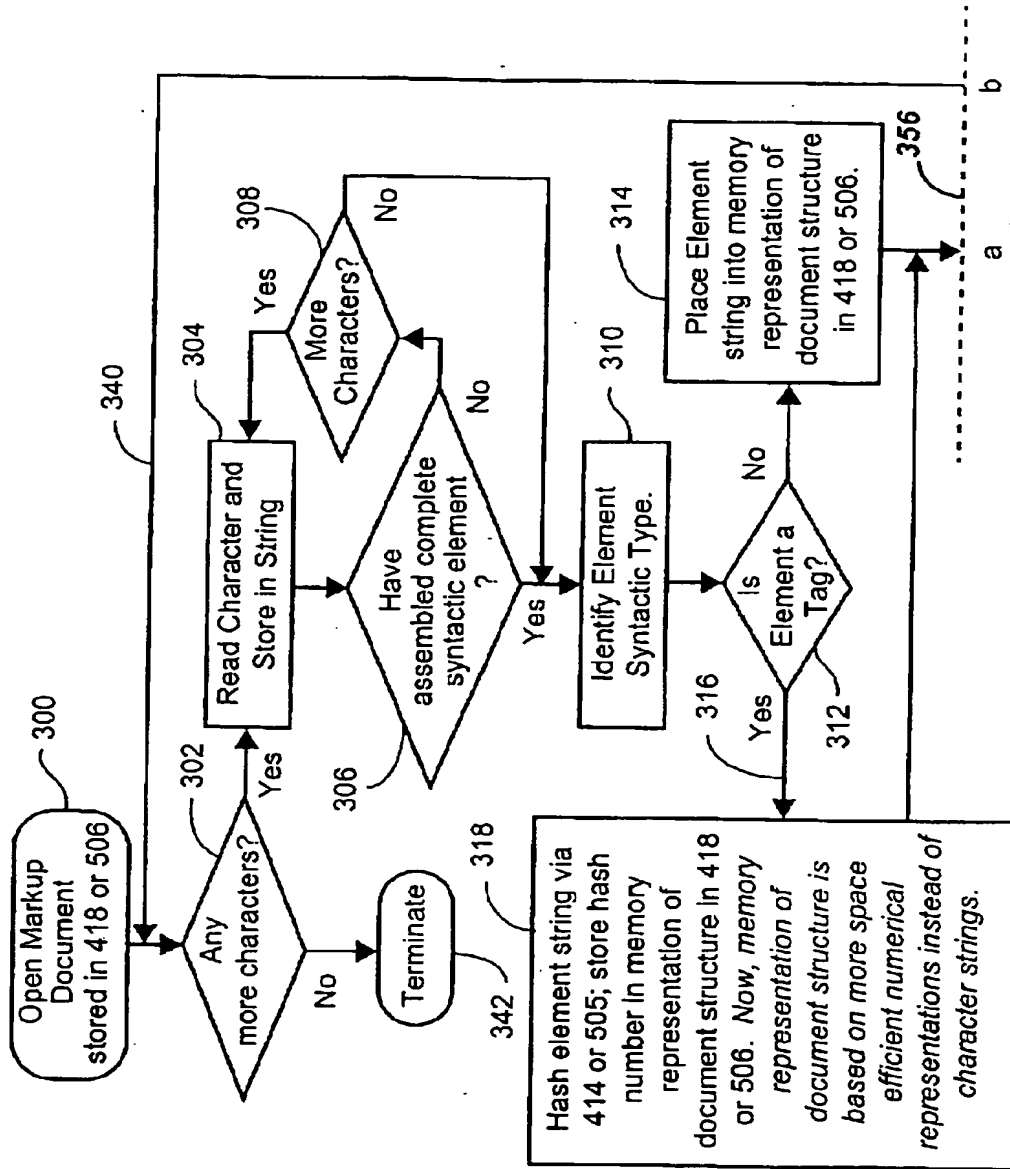


Fig. 3(a)

Fig. 3(a)

Fig. 3(b)

Fig. 3(c)

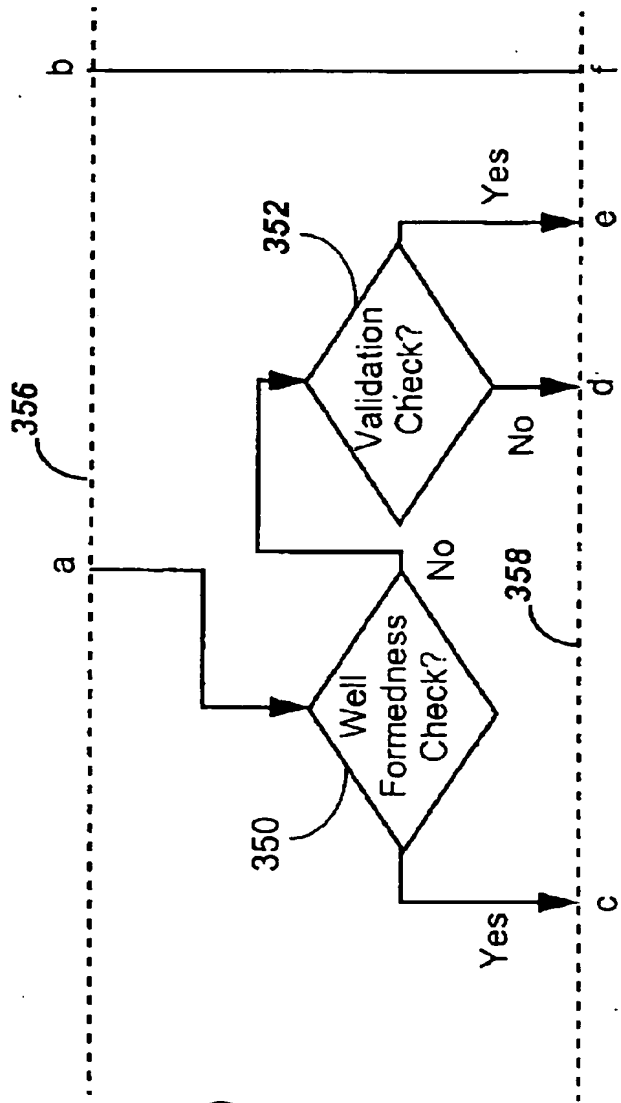
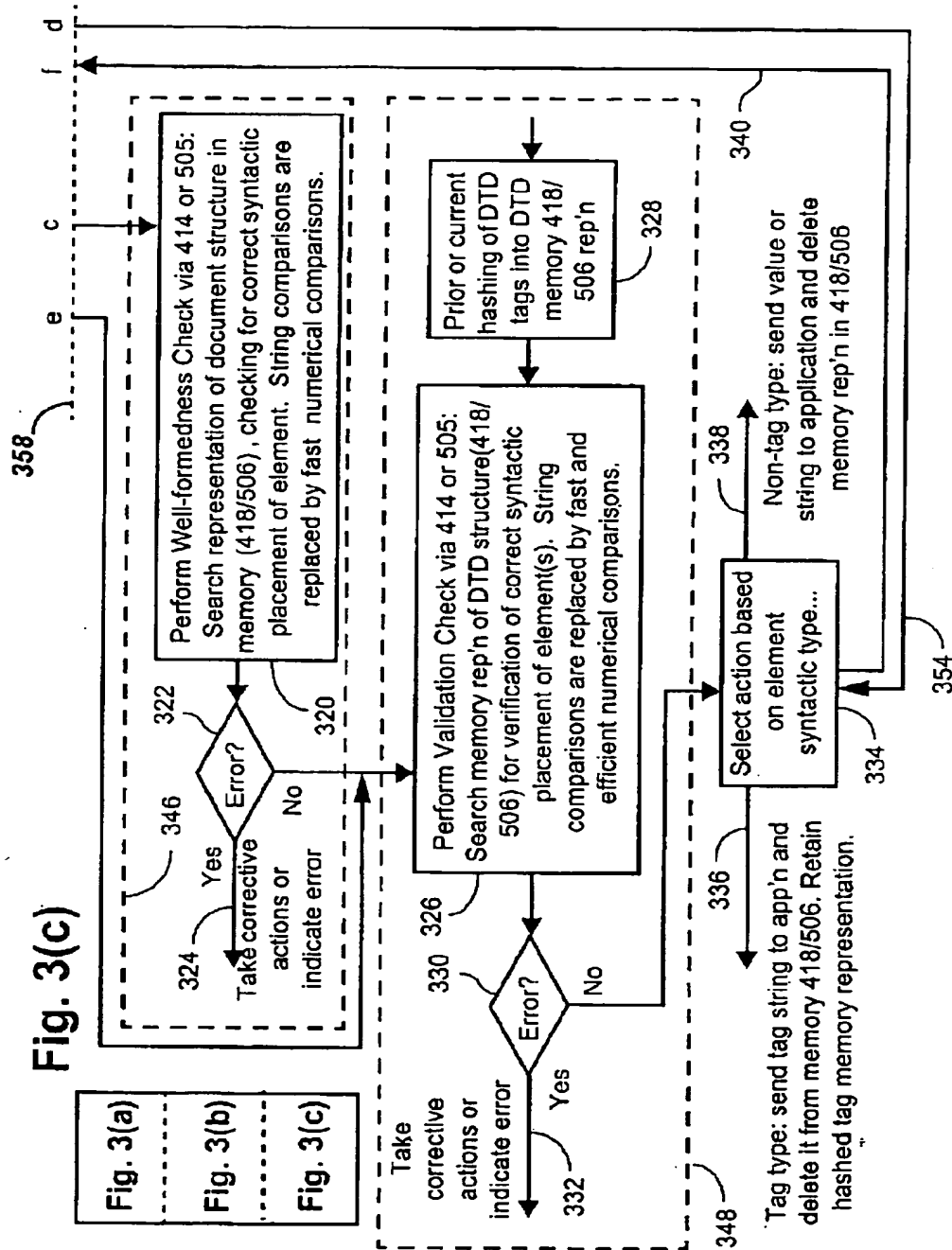
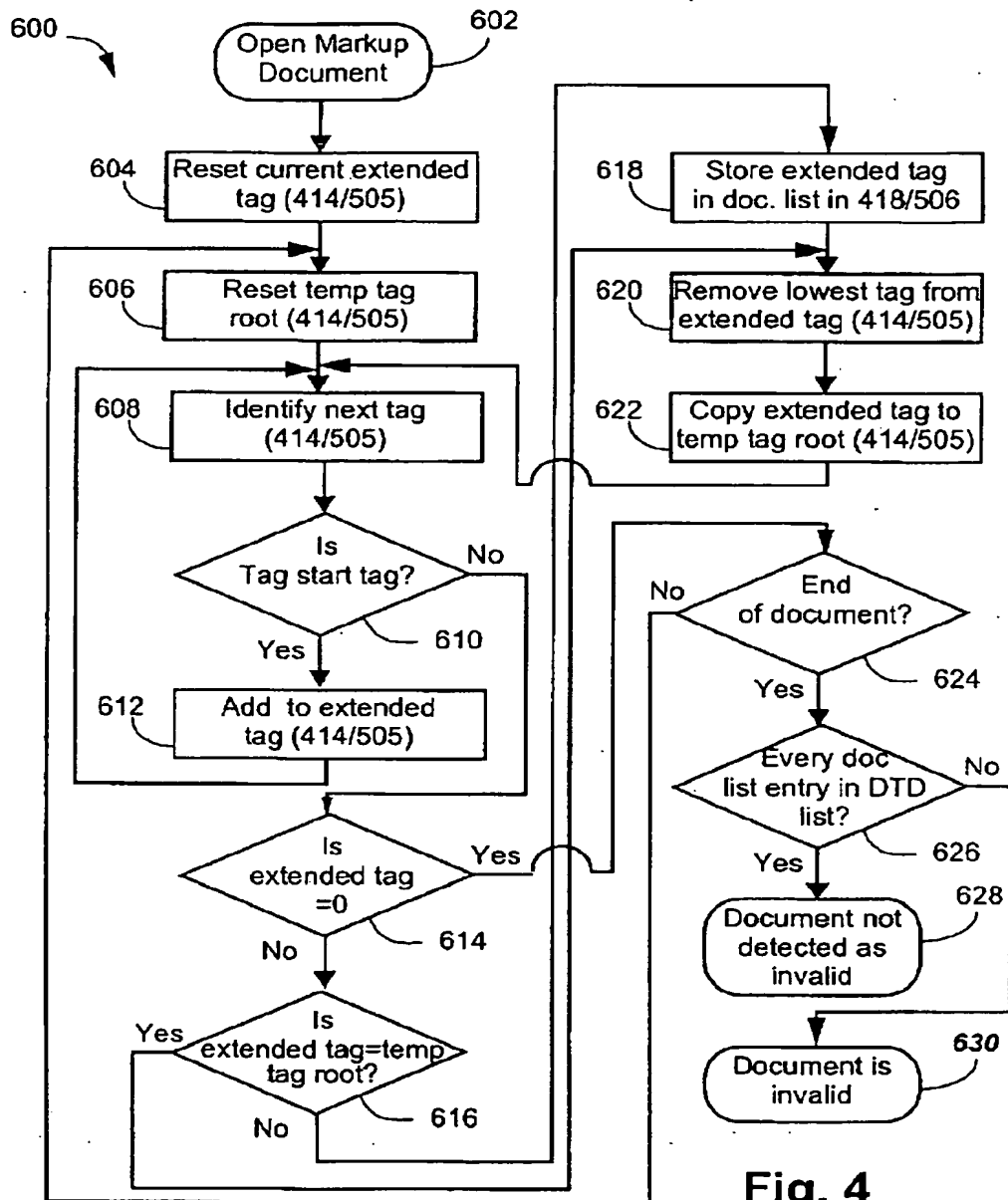
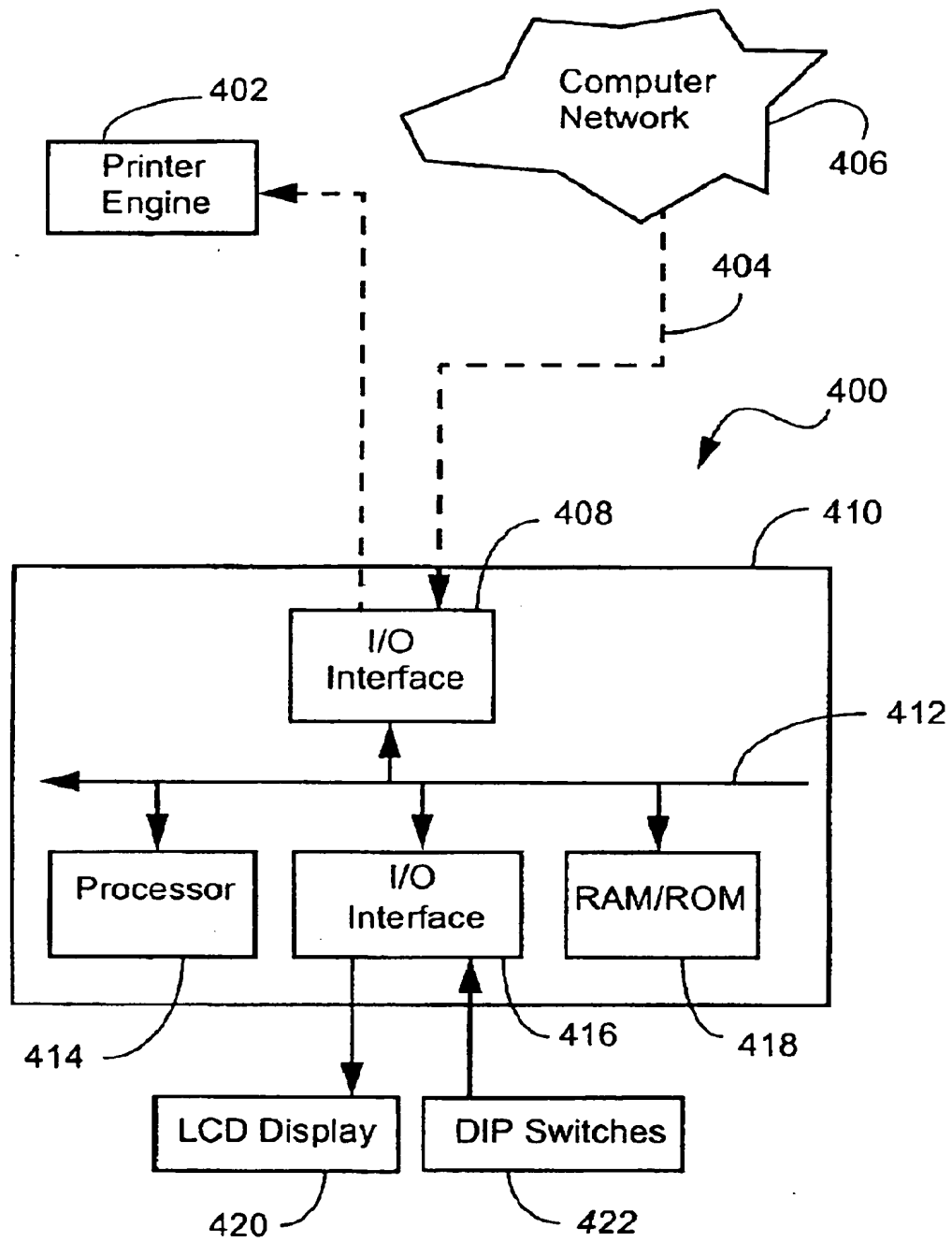


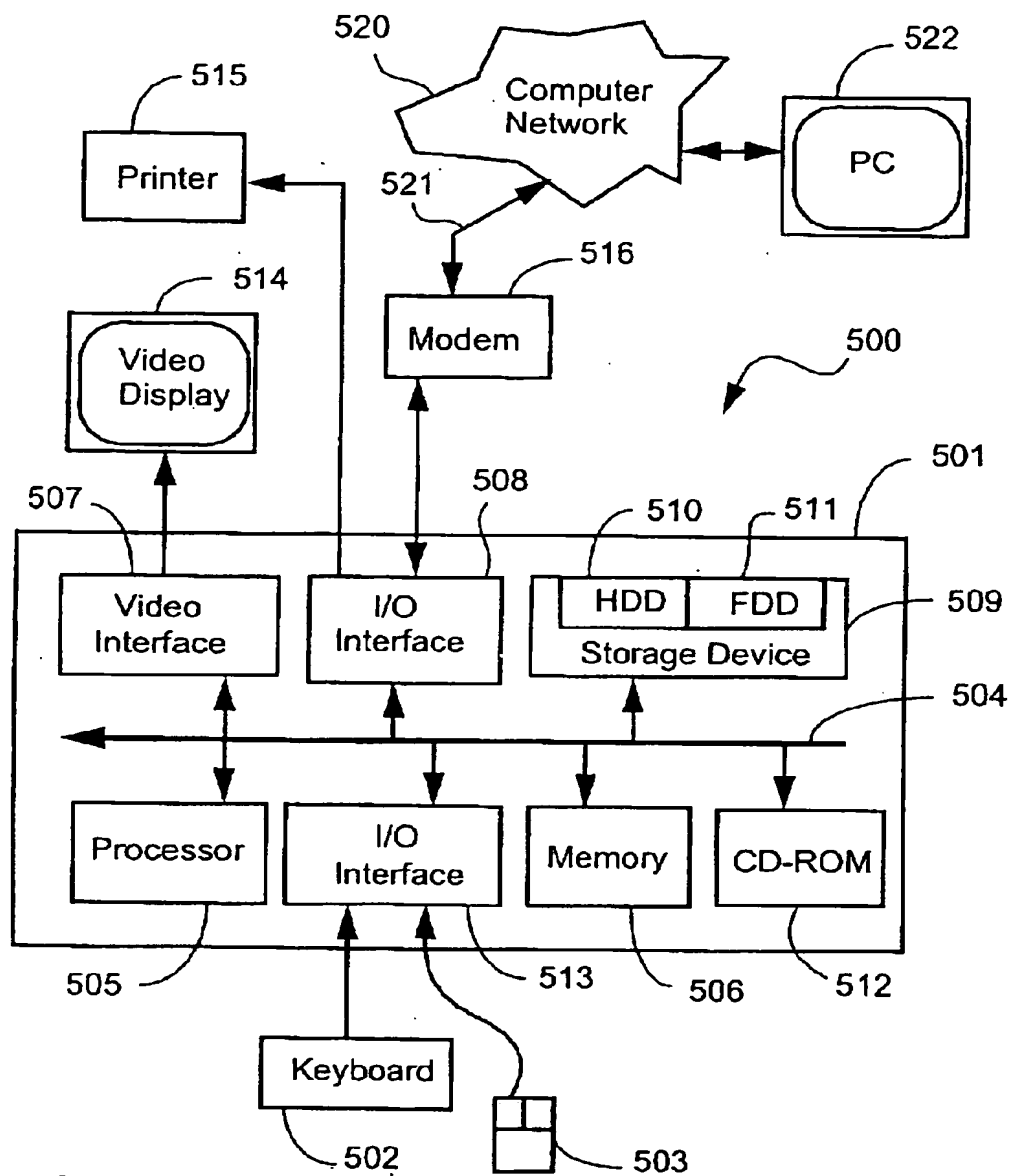
Fig. 3(b)

Fig. 3(a)		
Fig. 3(b)		
Fig. 3(c)		





**Fig. 5**

**Fig. 6**

1. Abstract

A method of parsing a markup language document comprising syntactic elements is disclosed, said method comprising, for one of said syntactic elements, the steps of identifying (310) a type of the element, processing (318) the element by determining a hash representation thereof if said type is a first type, and augmenting (314) an at least partial structural representation of the document using the hash representation if said type is said first type.

2. Representative Drawing

Fig.3(a)